

## Trabajo Fin de Grado

Control teleoperado de un sistema multi-robot  
para aplicaciones quirúrgicas

*Teleoperated Control of a multi-robot system for  
surgical applications*

Autor

Daniel Bescós León

Director

Gonzalo López Nicolás



## RESUMEN

En la actualidad, los ámbitos de la ingeniería y la medicina han estrechado sus relaciones gracias al desarrollo tecnológico de los últimos años. La unión de estas dos ramas de conocimiento busca aumentar la calidad de la asistencia médica y la calidad de vida de las personas. Este proyecto se enfoca en el auge de la robótica médica o biorrobótica, especialmente en materia de intervenciones quirúrgicas apoyadas con el uso de brazos robóticos, gracias a los cuales la recuperación de los pacientes es más rápida que mediante los métodos convencionales.

En este proyecto se realiza el estudio de varias aplicaciones quirúrgicas en la que los robots efectúan tareas propias de las mismas, como realizar incisiones, suturar, drenar o manipular material quirúrgico. Para este estudio se ha simulado una estación de trabajo en *RobotStudio* con la configuración existente en el laboratorio del Departamento de Informática e Ingeniería de Sistemas de la Universidad de Zaragoza, con el fin de que en un futuro estas aplicaciones puedan ser implementadas en dicho laboratorio.

En particular, el proyecto se fundamenta en el uso de dos brazos robóticos IRB120 de ABB que están disponibles en el laboratorio. Teniendo en cuenta la aplicación abordada, se ha efectuado un rediseño de las herramientas de los brazos robóticos disponibles en el laboratorio para que pudieran ser utilizadas con la distribución real de la estación de trabajo. También, se les ha dotado de una mayor funcionalidad (giro y apertura de pinza) que permitía una simulación de las mismas más parecida a las herramientas realmente utilizadas en una intervención quirúrgica.

Una de las tareas principales en este proyecto es la realización de la programación tanto de la interfaz gráfica GUI en *MATLAB* como de los brazos robóticos en *RAPID*. Es decir, el desarrollo del código necesario para que se lleven a cabo las diferentes aplicaciones abordadas.

Por lo tanto, para la mayoría de las aplicaciones se ha diseñado una interfaz gráfica de usuario GUI intuitiva en *MATLAB* para que el médico encargado de la intervención quirúrgica pueda controlar los brazos robóticos y supervisar su funcionamiento.

El control externo de los brazos robóticos a través de *MATLAB* se realiza mediante un socket de comunicación que permita la gestión simultánea de múltiples robots. El protocolo de comunicación utilizado es el TCP/IP, cuya comunicación está compuesta por un servidor y un cliente y permite el flujo bidireccional de información.

Finalmente, las aplicaciones desarrolladas han sido validadas en simulación mostrando un adecuado comportamiento y el código desarrollado en todas ellas permite asentar los fundamentos de futuras aplicaciones de interés.





## ABSTRACT

Recent technological developments have resulted in a strengthening of the relationship between the fields of engineering and medicine. The merging of these two areas of knowledge seeks to enhance both the quality of health care and the quality of life for people. This project focuses on the boom area of medical robotics or bio-robotics; particularly in terms of surgical procedures supported by robotic arms, thanks to which the recovery of patients is faster than it is with conventional methods.

In this project we have carried out a case study on several surgical applications including robots performing tasks such as incisions, sutures, draining or handling surgical material. For this study, a workstation has been simulated using *RobotStudio* within the existing configuration of the Department of Computing Science and Systems Engineering laboratory of the University of Zaragoza. The aim is to implement these applications in the laboratory in the future.

In particular, the project is based on the use of two ABB IRB120 robotic arms that are available in the laboratory. Considering the application addressed, we have carried out a redesign of the tools of the robotic arms available for use in the laboratory in order to use them in the current set-up of the workstation. Also, the tools have been equipped with greater functionalities (rotation and opening of forceps) that allow the simulation to be closer to the real tools used in a surgical procedure.

One of the main tasks in this project is the programming both the graphical user interface (GUI) in *MATLAB* and the robotic arms in *RAPID*. i.e. the development of the code necessary to produce an optimal behavior for the different applications addressed.

Thus, for most of the applications, an intuitive GUI has been designed in *MATLAB* so that the physician in charge of the surgical procedure can control the robotic arms and supervise their operation.

The external control of the robotic arms through *MATLAB* has been implemented by a communication socket that enables the simultaneous management of multiple robots. The communication protocol used is TCP/IP, which communicates between a server and a client, and permits bidirectional flow of information.

Finally, the developed applications have been validated as showing correct behaviour within simulations. These applications' code will provide the groundwork for future applications of interest.



# ÍNDICE

<b>RESUMEN</b> .....	<b>I</b>
<b>ABSTRACT</b> .....	<b>III</b>
<b>LISTA DE TABLAS</b> .....	<b>VII</b>
<b>LISTA DE FIGURAS</b> .....	<b>VII</b>
<b>CAPÍTULO 1. INTRODUCCIÓN.</b> .....	<b>1</b>
1.1. OBJETIVOS DEL PROYECTO. ....	1
1.2. ESTRUCTURA DE LA MEMORIA. ....	2
1.3. ESTADO DE LA MATERIA. ....	3
<b>CAPÍTULO 2. HERRAMIENTAS UTILIZADAS</b> .....	<b>6</b>
2.1. BRAZO ROBÓTICO IRB 120.....	6
2.2. CONTROLADOR IRC5. ....	7
2.3. ROBOTSTUDIO.....	7
2.4. LENGUAJE DE PROGRAMACIÓN RAPID.....	8
2.5. MATLAB.....	9
<b>CAPÍTULO 3. DISEÑO Y CONFIGURACIÓN DE LA ESTACIÓN DE TRABAJO EN ROBOTSTUDIO</b> .....	<b>10</b>
3.1. DISEÑO GENERAL DE LA ESTACIÓN.....	10
3.2. CONFIGURACIÓN DE LOS CONTROLADORES VIRTUALES.....	11
3.3. DISEÑO DE LAS HERRAMIENTAS DE TRABAJO. ....	12
3.4. CONFIGURACIÓN DE LAS HERRAMIENTAS DE TRABAJO. ....	13
3.5. CONFIGURACIÓN DE FUNCIONALIDAD DE LA HERRAMIENTA DE SUTURA.....	16
<b>CAPÍTULO 4. PROTOCOLO DE COMUNICACIÓN.</b> .....	<b>22</b>
4.1. PROTOCOLO TCP/IP.....	22
4.2. ESTABLECIMIENTO DE LA COMUNICACIÓN. ....	23
4.3. TRANSMISIÓN DE INFORMACIÓN. ....	25
4.3.1. <i>Emisión de datos desde MATLAB.</i> .....	25
4.3.2. <i>Recepción de datos desde RobotStudio.</i> .....	25
4.3.3. <i>Emisión de datos desde RobotStudio</i> .....	25
4.3.4. <i>Recepción de datos desde MATLAB</i> .....	26
4.3.5. <i>Transmisión de posiciones del robot.</i> .....	26
4.4. CIERRE DE LA CONEXIÓN. ....	28
<b>CAPÍTULO 5. MODELADO CINEMÁTICO DEL ROBOT IRB 120</b> .....	<b>29</b>
5.1. MODELO GEOMÉTRICO DIRECTO DE POSICIÓN: FUNDAMENTO TEÓRICO.....	29
5.2. MODELO GEOMÉTRICO DIRECTO DE POSICIÓN DEL ROBOT IRB 120. ....	33
5.3. CUATERNIOS. ....	37
5.4. MATRIZ DE CAMBIO DE BASE. ....	38
<b>CAPÍTULO 6. DESARROLLO DE UNA INTERFAZ GRÁFICA DE CONTROL EN MATLAB. GUI</b> .....	<b>39</b>
6.1. CONCEPTOS GENERALES. ....	39
6.2. CASOS DE ESTUDIO. ....	41
<b>CAPÍTULO 7. CONTROL AUTOMÁTICO DE OPERACIÓN INCISIÓN-SUTURA</b> .....	<b>43</b>
7.1. OPERACIÓN DE INCISIÓN. ....	43
7.1.1. <i>Conexión con el robot.</i> .....	44
7.1.2. <i>Mover el robot a la posición para empezar la incisión.</i> .....	45
7.1.3. <i>Actualizar posición de la herramienta.</i> .....	47
7.1.4. <i>Incisión</i> .....	48
7.1.5. <i>Funciones independientes</i> .....	50
7.2. OPERACIÓN DE SUTURA.....	51

7.2.1.	Conexión con el robot.....	51
7.2.2.	Mover robot a posición cercana al cuerpo humano.....	51
7.2.3.	Suturar.....	53
7.2.4.	Funciones independientes.....	55
7.3.	ESQUEMA DE LAS ETAPAS DE LA SIMULACIÓN.....	55
<b>CAPÍTULO 8. CONTROL MANUAL DE OPERACIÓN INCISIÓN-SUTURA.....</b>		<b>57</b>
8.1.	OPERACIÓN DE INCISIÓN.....	57
8.1.1.	Configuración del movimiento.....	58
8.1.2.	Mover Robot en coordenadas articulares y cartesianas.....	59
8.1.3.	Guardar posiciones.....	60
8.1.4.	Incisión.....	61
8.2.	OPERACIÓN DE SUTURA.....	62
8.2.1.	Funcionalidad de la pinza.....	62
8.3.	ESQUEMA DE LAS ETAPAS DE LA SIMULACIÓN.....	65
<b>CAPÍTULO 9. CONTROL SIMULTÁNEO DE OPERACIÓN DRENAJE-SUTURA.....</b>		<b>66</b>
9.1.	DATOS DE LA INTERVENCIÓN.....	67
9.2.	MOVER SINCRONIZADAMENTE.....	68
9.3.	ESQUEMA DE LAS ETAPAS DE LA SIMULACIÓN.....	70
<b>CAPÍTULO 10. MANIPULACIÓN DE MATERIAL E INSTRUMENTAL QUIRÚRGICO.....</b>		<b>71</b>
<b>CAPÍTULO 11. CONCLUSIONES Y LÍNEAS FUTURAS.....</b>		<b>76</b>
<b>BIBLIOGRAFÍA.....</b>		<b>78</b>
<b>ANEXO I. PLANOS DE LA HERRAMIENTA.....</b>		<b>80</b>
<b>ANEXO II. MANUAL DE USUARIO.....</b>		<b>82</b>
<b>ANEXO III. SCRIPTS DE MATLAB.....</b>		<b>84</b>
<b>ANEXO IV. SCRIPTS DE RAPID ROBOTSTUDIO.....</b>		<b>92</b>

## LISTA DE TABLAS

TABLA 1.1. LEYES DE LA ROBÓTICA .....	3
TABLA 5.1. PARÁMETROS DENAVIT-HARTENBERG DEL ROBOT IRB 120.....	35
TABLA 7.1. ESQUEMA ETAPAS SIMULACIÓN ESCENARIO I .....	56
TABLA 8.1. ESQUEMA ETAPAS SIMULACIÓN ESCENARIO II .....	65
TABLA 9.1. ESQUEMA ETAPAS SIMULACIÓN ESCENARIO III .....	70
TABLA 10.1. ESQUEMA ETAPAS SIMULACIÓN ESCENARIO IV .....	75

## LISTA DE FIGURAS

FIGURA 1.1. SISTEMA ROBÓTICO DA VINCI.....	5
FIGURA 2.1. EJES Y POSICIONES DE TRABAJO DEL ROBOT IRB 120 .....	6
FIGURA 2.2. ÁREA DE TRABAJO DEL CENTRO DE LA MUÑECA (EJE 5) DEL ROBOT IRB 120 (MEDIDAS EN MM) .....	6
FIGURA 2.3. CONTROLADOR IRC5 Y FLEXPENDANT .....	7
FIGURA 2.4. INTERFAZ DE ROBOTSTUDIO .....	8
FIGURA 2.5. ESTRUCTURA DE APLICACIÓN RAPID.....	8
FIGURA 2.6. INTERFAZ DE MATLAB.....	9
FIGURA 3.1. PLANO DE LA ESTACIÓN DE TRABAJO .....	10
FIGURA 3.2. CONFIGURACIÓN DEL CONTROLADOR VIRTUAL .....	11
FIGURA 3.3. RENDERIZADO DE LA HERRAMIENTA .....	12
FIGURA 3.4. HERRAMIENTAS DE INCISIÓN Y DE SUTURA .....	13
FIGURA 3.5. PROCESO DE ESCALADO DE LAS HERRAMIENTAS .....	14
FIGURA 3.6. CREACIÓN DE HERRAMIENTA .....	15
FIGURA 3.7. PASOS PARA LA CREACIÓN DE LA HERRAMIENTA.....	15
FIGURA 3.8. FIJACIÓN DE LA HERRAMIENTA EN EL ROBOT .....	16
FIGURA 3.9. IMPORTACIÓN HERRAMIENTA SEPARADA EN DIFERENTES PARTES.....	17
FIGURA 3.10. CREACIÓN DE MECANISMO PARA FUNCIONALIDAD DE HERRAMIENTA .....	17
FIGURA 3.11. CREACIÓN DE ESLABONES Y ARTICULACIONES .....	18
FIGURA 3.12. POSICIONES DE LA HERRAMIENTA .....	19
FIGURA 3.13. TIEMPOS DE TRANSICIÓN ENTRE POSICIONES.....	19
FIGURA 3.14. LÓGICA DE LA FUNCIONALIDAD DE LA HERRAMIENTA .....	20
FIGURA 3.15. ESTACIÓN DE TRABAJO FINAL Y POSICIÓN INICIAL DE LOS ROBOTS .....	21
FIGURA 4.1. ESTRUCTURA CLIENTE-SERVIDOR .....	23
FIGURA 4.2. ESTABLECIMIENTO DE LAS VARIABLES EN ROBOTSTUDIO .....	23
FIGURA 4.3. ESTABLECIMIENTO DE LA CONEXIÓN EN ROBOTSTUDIO .....	24
FIGURA 4.4. ESTABLECIMIENTO DE LA CONEXIÓN EN MATLAB .....	24
FIGURA 4.5. EJEMPLO EMISIÓN DE DATOS DESDE MATLAB .....	25
FIGURA 4.6. EJEMPLO RECEPCIÓN DE DATOS DESDE ROBOTSTUDIO .....	25
FIGURA 4.7. EJEMPLO EMISIÓN DE DATOS DESDE ROBOTSTUDIO.....	26
FIGURA 4.8. EJEMPLO RECEPCIÓN DE DATOS DESDE MATLAB .....	26
FIGURA 4.9. TRANSMISIÓN DE POSICIONES DE ROBOTSTUDIO A MATLAB.....	27
FIGURA 4.10. TRANSMISIÓN DE POSICIONES DE MATLAB A ROBOTSTUDIO .....	28
FIGURA 4.11. ESTABLECIMIENTO Y CIERRE DE LA CONEXIÓN .....	28
FIGURA 5.1. MATRIZ DE TRANSFORMACIÓN HOMOGÉNEA .....	29
FIGURA 5.2. SITUACIÓN DEL ORIGEN DE LOS SISTEMAS DE COORDENADAS SEGÚN D-H .....	30
FIGURA 5.3. SITUACIÓN DEL VECTOR X DE LOS SISTEMAS DE COORDENADAS SEGÚN D-H .....	31
FIGURA 5.4. PARÁMETROS DENAVIT-HARTENBERG .....	32
FIGURA 5.5. DIMENSIONES DEL ROBOT IRB 120.....	33
FIGURA 5.6. ESLABONES Y ARTICULACIONES DEL ROBOT IRB 120.....	33
FIGURA 5.7. SISTEMAS COORDENADOS DEL ROBOT IRB 120 SEGÚN DENAVIT-HARTENBERG .....	34
FIGURA 5.8. MATRICES DE TRANSFORMACIÓN HOMOGÉNEA ENTRE SISTEMAS DE COORDENADAS .....	35
FIGURA 5.9. CAMBIO DE ORIENTACIÓN DE LA HERRAMIENTA .....	37
FIGURA 6.1. FUNCIÓN DE INICIALIZACIÓN DE LA GUI.....	39
FIGURA 6.2. FUNCIÓN DE APERTURA Y SALIDA DE LA GUI.....	40
FIGURA 6.3. EJEMPLO USO TAG ESTRUCTURA HANDLES Y FUNCIÓN CALLBACK.....	41
FIGURA 7.1. ESCENARIO I: INTERFAZ GRÁFICA DE LA OPERACIÓN DE INCISIÓN .....	43

<b>FIGURA 7.2.</b> PANEL: CONEXIÓN CON ROBOT.....	44
<b>FIGURA 7.3.</b> FUNCIONES PARA OBTENER LA DIRECCIÓN IP .....	44
<b>FIGURA 7.4.</b> FUNCIÓN BOTÓN CONECTAR .....	45
<b>FIGURA 7.5.</b> PANEL: MOVER ROBOT A POSICIÓN PARA EMPEZAR INCISIÓN .....	45
<b>FIGURA 7.6.</b> FRAGMENTO DE LA FUNCIÓN PANEL MOVER ROBOT A POSICIÓN PARA EMPEZAR INCISIÓN .....	47
<b>FIGURA 7.7.</b> PANEL: ACTUALIZAR POSICIÓN DE LA HERRAMIENTA .....	47
<b>FIGURA 7.8.</b> FUNCIÓN PANEL ACTUALIZAR POSICIÓN DE LA HERRAMIENTA.....	47
<b>FIGURA 7.9.</b> PANEL: INCISIÓN .....	48
<b>FIGURA 7.10.</b> FRAGMENTO DE LA FUNCIÓN INCISIÓN .....	49
<b>FIGURA 7.11.</b> FUNCIÓN CALLBACK BOTONES 'STOP' E 'IR A INTERFAZ DE SUTURAR' .....	50
<b>FIGURA 7.12.</b> ESCENARIO I: INTERFAZ GRÁFICA DE LA OPERACIÓN DE SUTURA .....	51
<b>FIGURA 7.13.</b> PANEL: MOVER ROBOT A POSICIÓN CERCANA AL CUERPO HUMANO .....	51
<b>FIGURA 7.14.</b> CUADRANTES DE CONFIGURACIÓN PARA EL EJE 1, 4 Y 6 DE UN ROBOT DE 6 EJES .....	52
<b>FIGURA 7.15.</b> PANEL: SUTURAR .....	53
<b>FIGURA 8.1.</b> ESCENARIO II: INTERFAZ GRÁFICA DE LA OPERACIÓN DE INCISIÓN .....	57
<b>FIGURA 8.2.</b> PANEL: CONFIGURACIÓN DEL MOVIMIENTO .....	58
<b>FIGURA 8.3.</b> FUNCIÓN ELEGIR TIPO DE MOVIMIENTO .....	58
<b>FIGURA 8.4.</b> FUNCIÓN ELEGIR VELOCIDAD .....	59
<b>FIGURA 8.5.</b> PANEL: MOVER ROBOT EN COORDENADAS ARTICULARES/CARTESIANAS .....	59
<b>FIGURA 8.6.</b> FUNCIÓN BOTÓN PARA MOVER AL ROBOT EN EL EJE X POSITIVO .....	60
<b>FIGURA 8.7.</b> PANEL: GUARDAR POSICIONES.....	61
<b>FIGURA 8.8.</b> PANEL: INCISIÓN .....	61
<b>FIGURA 8.9.</b> ESCENARIO II: INTERFAZ GRÁFICA DE LA OPERACIÓN DE SUTURA .....	62
<b>FIGURA 8.10.</b> PANEL: FUNCIONALIDAD DE LA PINZA .....	62
<b>FIGURA 8.11.</b> LÓGICA DE ESTACIÓN ESCENARIO II .....	63
<b>FIGURA 8.12.</b> RAPID: FUNCIONALIDAD HERRAMIENTA ESCENARIO II .....	64
<b>FIGURA 9.1.</b> ESCENARIO III: INTERFAZ GRÁFICA DE LA OPERACIÓN DRENAJE-SUTURA .....	66
<b>FIGURA 9.2.</b> PANEL: DATOS DE LA INTERVENCIÓN .....	67
<b>FIGURA 9.3.</b> LÓGICA DE ESTACIÓN ESCENARIO III .....	68
<b>FIGURA 9.4.</b> RAPID: FUNCIONALIDAD DE LA HERRAMIENTA ESCENARIO III.....	69
<b>FIGURA 10.1.</b> ESTACIÓN DE TRABAJO ESCENARIO IV .....	71
<b>FIGURA 10.2.</b> COMPONENTE INTELIGENTE HERRAMIENTA ESCENARIO IV .....	72
<b>FIGURA 10.3.</b> LÓGICA DE ESTACIÓN ESCENARIO IV .....	74

# CAPÍTULO 1. INTRODUCCIÓN.

---

## 1.1. Objetivos del proyecto.

---

El objetivo final de este proyecto es la automatización robótica de procedimientos que puedan darse en una intervención quirúrgica real, específicamente la acción de realizar una incisión y la consecuente sutura de la misma.

Inicialmente, se desarrollará el diseño, configuración y programación de una estación de trabajo simulada con ayuda del software *RobotStudio*. Dicha estación simulará un quirófano y será controlada por otro software denominado *MATLAB* a través de un flujo de información gracias a un protocolo de comunicación TCP/IP.

Finalmente, debido a una posible futura implementación de este proyecto en la vida real, se diseñarán unas interfaces gráficas de usuario GUI (Graphical User Interface) básicas e intuitivas debido a que la persona que vaya a hacer uso de ellas, con toda probabilidad, será un médico del que no se puede presuponer ningún tipo de conocimiento en el ámbito de la robótica.

El diseño diferentes interfaces gráficas debido a que se estudian diferentes aplicaciones quirúrgicas multi-robot como: operación incisión-sutura, operación drenaje-sutura y manipulación de material e instrumental quirúrgico entre robots. Además, algunas de estas aplicaciones se dotarán de la capacidad de que los robots puedan interactuar y coordinarse entre ellos.

Asimismo, como el laboratorio de la Universidad dispone de los medios necesarios y el proyecto se ha orientado a la distribución de los robots del mismo, se prevé que en el futuro y como continuación de este trabajo se estudie la implementación de las aplicaciones utilizando un tejido que pueda simular la piel humana y herramientas que simulen a las precisas para realizar la incisión y la sutura (bisturí y grapadora médica).

Cabe destacar el Trabajo Fin de Grado de Eduardo Renta García [1], que consiste en el uso de la tecnología robótica para lograr la reproducción de imágenes digitales en papel. Del mismo, ha resultado útil para este proyecto tanto el establecimiento de comunicación mediante el protocolo TCP/IP como alguna sub-función del código desarrollado para transmitir información. Es preciso aclarar que la aplicación de este trabajo es completamente diferente y, por consecuente, la programación del mismo también. Es decir, el código que se ha podido aprovechar es una parte mínima del conjunto de la programación.

## 1.2. Estructura de la memoria.

---

A continuación, se va a explicar de forma muy general la manera en que se ha estructurado la memoria de este Trabajo Fin de Grado.

- El **capítulo 1** se corresponde con el actual y en él se introduce el tema elegido para este proyecto, así como los objetivos y la estructura interna del mismo.
- En el **capítulo 2** se describen cada una de las herramientas más importantes utilizadas en la elaboración de este proyecto: los modelos de los robots y sus controladores y los software empleados.
- En el **capítulo 3** se desarrolla el procedimiento de diseño y configuración de todos los elementos dispuestos en la estación de trabajo.
- En el **capítulo 4** se explica en qué consiste el protocolo de comunicación seleccionado, el por qué se ha seleccionado ese en concreto y cómo se establece y cierra la conexión al igual que la transmisión de datos.
- En el **capítulo 5** se explican las bases del modelo geométrico general y se aplica al robot de estudio, además de algunos conceptos teóricos importantes para poder entender el proyecto con claridad.
- En el **capítulo 6** se describen las bases de la creación de una interfaz gráfica GUI en *MATLAB* para abordar el control de los robots de las diferentes aplicaciones a través de ella y se introducen brevemente cada una de estas aplicaciones de estudio.
- En los **capítulos 7, 8, 9 y 10** se desarrolla el proceso de programación de cada uno de los cuatro escenarios de las aplicaciones de estudio y el proceso de creación de la interfaz gráfica GUI en los escenarios que la posean.
- En el **capítulo 11** se comentan las conclusiones alcanzadas con los resultados del proyecto y cuales son las vías con las que seguir desarrollando este tema en un futuro.
- Después se encuentra la bibliografía, donde se enumeran las fuentes consultadas como apoyo para la elaboración del proyecto.

Finalmente están los Anexos, en los cuales se detallan los planos de la geometría de las herramientas, un manual de usuario y los scripts de la programación de *MATLAB* y *RobotStudio*.



### 1.3. Estado de la materia.

En los últimos años, gracias al avance de la tecnología, los ámbitos de la medicina y la ingeniería han estrechado lazos importantes. Este hecho dio lugar a la formación de una nueva rama de conocimiento denominada Ingeniería Biomédica. Actualmente, la ingeniería biomédica combina la experiencia de la ingeniería con las necesidades médicas para obtener beneficios en el cuidado de la salud [2]. Además, actualmente ya no se concibe la medicina sin la tecnología, y gracias al desarrollo de la misma han surgido varios avances como pueden ser los exoesqueletos capaces de realizar la función de movimiento que una persona no puede realizar por sí misma, la impresión 3D de órganos artificiales o la cirugía con la ayuda de robots [3]. El último es, precisamente, en el que pretende enfatizar el presente Trabajo Fin de Grado.

Para entender cómo la sociedad actual ha llegado a plantearse el uso de robots para cirugía es conveniente hacer un repaso de los términos **robot** y **robótica** y su evolución a lo largo de los años.

El término **robot** surgió de la palabra eslava “robota”, la cual significa trabajo forzado, y fue acuñado por primera vez por el escritor de nacionalidad checa Karel Capek en el año 1921 tras el estreno de su obra *Rossum’s Universal Robots (R.U.R.)* en el Teatro Nacional de Praga. Debido al gran éxito y aceptación que tuvo la obra en su época, otros escritores del género literario de la ciencia ficción comenzaron a utilizar este término consiguiendo que no quedara en desuso [4].

Más adelante, Isaac Asimov publicó en 1950 un libro de relatos cortos reunidos llamado *I Robot* en el que el autor utilizaba por primera vez el término **robótica**. En el relato, ambientado en un futuro distópico, se postulaban las tres leyes de la robótica que se han convertido en un referente, al menos, teórico [5].

1	Un robot no debe dañar a un ser humano ni, por su pasividad, dejar que un ser humano sufra daño.
2	Un robot debe obedecer las órdenes que le son dadas por un ser humano, excepto cuando estas órdenes están en oposición con la primera Ley.
3	Un robot debe proteger a su propia existencia, hasta donde esta protección no esté en conflicto con la primera o segunda ley.

*Tabla 1.1. Leyes de la robótica*

El auge de la robótica moderna se da durante la primera mitad del siglo XX debido a [6]:

- El inicio del desarrollo de la ingeniería en diferentes ramas, como por ejemplo: mecánica, electrónica, informática, etc.
- El progreso en el estudio de las matemáticas y la física teórica (Lagrange, Newton y Euler) con el que se consigue, posteriormente, desarrollar las ecuaciones que explican la dinámica de los robots actuales.
- Los avances en computación que permiten desarrollar máquinas muy cercanas al ideal de automatismo y autonomía.

Inicialmente, la robótica surge de la necesidad continua que los humanos tienen de buscar alternativas que faciliten y optimicen cualquier tarea productiva. De esta necesidad surge el primer robot industrial en 1938, creado por H. Roselund y W. Pollard, que consistía en un brazo articulado cuya función era pintar con spray. A partir de dicho acontecimiento se produce un importante despegue en la robótica, destacando la década de los 50 en la que se desarrollan los primeros robots [7]. Sin embargo, no fue hasta 1954 cuando George Devol patentó la idea del primer robot programable (Unimate), estableciendo las bases del robot industrial moderno [8].

Años más tarde, en la década de los 90, debido al trabajo repetitivo que los operarios de las fábricas realizaban, surgió la necesidad de buscar soluciones automatizadas que tuvieran el potencial suficiente para resolver este problema. Debido a esta búsqueda de mejorar la ergonomía de los trabajadores se creó el primer robot de tipo colaborativo en la industria. A pesar de que esta tendencia empezó a emerger en la década de los 90, no fue hasta el año 2008 cuando la empresa Universal Robots vendió el primer robot colaborativo [9].

Los **robots colaborativos**, comúnmente conocidos como cobots, son robots creados con la finalidad de interactuar con los operarios en las fábricas de manera que la producción consiga unir las ventajas que aportan cada uno de ellos, como pueden ser la racionalidad humana y la precisión de los robots. Cabe destacar el aumento en la productividad en el caso del uso de cobots, ya que según un estudio realizado en 2016 por investigadores del MIT (Massachusetts Institute of Technology), la colaboración entre humanos y robots es un 85% más productiva [10].

En lo relacionado con la medicina, los robots se introducen en esta rama de conocimiento principalmente debido a la elevada precisión que son capaces de aportar [11]. La historia moderna de la cirugía robótica comienza en 1985 con el robot Puma 560, utilizado para realizar biopsias neuroquirúrgicas mediante punción (laparoscopia). Con el paso de los años, se ha observado que el área donde ha tenido mayor impacto ha sido en Urología, en la cual se realizan la mayor parte de las intervenciones quirúrgicas robóticas mínimamente invasivas actualmente [12]. De hecho, en 2009 el número de prostatectomías robóticas para el tratamiento del cáncer de próstata realizadas superaba la cifra de 60.000 [13].

Una cirugía mínimamente invasiva consiste en realizar incisiones pequeñas e introducir un laparoscopio, es decir, cámaras tubulares diminutas que proporcionan visión del interior del cuerpo para guiar la operación. Las ventajas de este tipo de intervención son la menor pérdida de sangre debido a las pequeñas incisiones producidas, por lo que hay una disminución de la necesidad de transfusiones de sangre. Además, permite una recuperación más indolora, lo que se traduce en una estancia hospitalaria más breve y, uno de los aspectos más importantes es el mejor aspecto físico por las cicatrices tan pequeñas que se manifiestan después de la operación [14].

Finalmente, el sistema robótico más evolucionado de la **cirugía robótica** mínimamente invasiva se lanzó al mercado en 1999 y es conocido como *Da Vinci* (ver Figura 1.1). Este robot es controlado manualmente por un cirujano que opera desde una consola y requiere, en todos los casos, la intervención y toma de decisiones de un profesional. El robot *Da Vinci* permite optimizar el rango de acción de la mano humana, reduciendo el posible temblor y perfeccionando todos los movimientos del cirujano. El robot se compone de una consola ergonómica desde la que el cirujano opera (normalmente se encuentra en el mismo quirófano), una torre de visión y el robot esclavo que posee tres o cuatro brazos robóticos interactivos controlados desde la consola [15]. Como medida de protección, el software está diseñado de manera que si el cirujano hace un movimiento brusco los brazos robóticos se frenan automáticamente. Además, mientras se está operando, el cirujano debe estar mirando por un sistema binocular, del cual si se aparta la mirada, un sistema de rayos infrarrojos desactiva los brazos [16].



Figura 1.1. Sistema robótico *Da Vinci*

Otro término destacable es el de **telecirugía**, es decir, utilizar la consola del sistema robótico para controlar un robot que no está físicamente en el mismo espacio que el cirujano. Este término surgió en la década de los 70 debido a que la NASA planteó operar a los astronautas a distancia. Actualmente, hay algunas referencias de esta técnica, aunque no son muchas. En 2001, cirujanos de Nueva York realizaron la primera operación transatlántica con un paciente de Francia. También, en 2006 se utilizó un robot para suturar a un paciente que vivía en la base submarina Aquarius con el objetivo de simular una telecirugía en el espacio. El principal problema que tenía este método de cirugía era el retraso en la comunicación, pero actualmente, debido a la mejora de las conexiones a Internet por cable, este problema ha desaparecido [17]. Además, la presente investigación de redes de comunicaciones 5G supondrá un avance muy importante en este ámbito [18].

## CAPÍTULO 2. HERRAMIENTAS UTILIZADAS.

En este capítulo se van a introducir las diferentes herramientas utilizadas para la realización de este proyecto. En éstas se incluye tanto el software, es decir, los programas utilizados, como el hardware, en este caso, el brazo articulado y su correspondiente controlador.

### 2.1. Brazo robótico IRB 120.

Este brazo robótico está diseñado por la reconocida corporación multinacional de generación de energía eléctrica y automatización industrial llamada *ABB*. Se trata del robot industrial más pequeño de *ABB* cuyas características más importantes son su peso, su capacidad de manejo y su alcance, que son 25 kg, 3 kg y 580 mm respectivamente.

Gracias a sus 6 ejes, este robot proporciona una solución ágil, compacta y ligera con un control superior y una precisión de trayectoria en cualquier aplicación. Además, se puede instalar en cualquier posición sin restricción alguna (ver Figura 2.1).

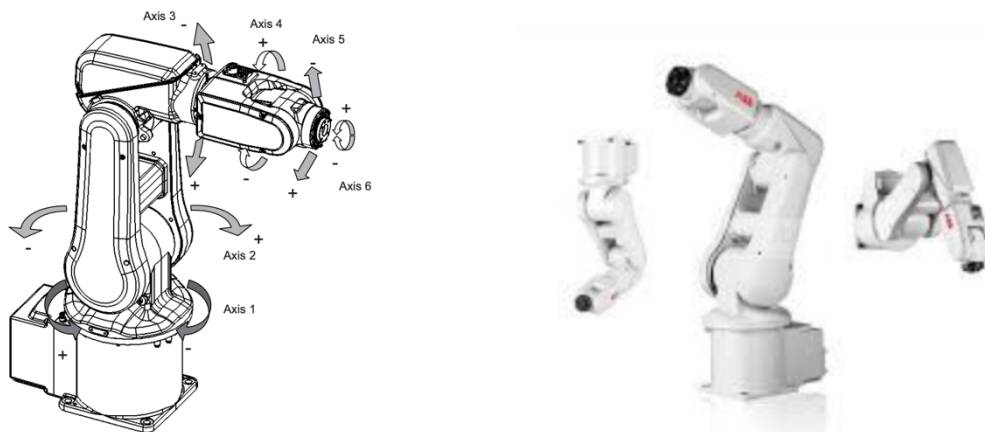


Figura 2.1. Ejes y posiciones de trabajo del robot IRB 120

A pesar de ser un robot diseñado para generar una gran capacidad de producción industrial de manera fiable, rentable y con una baja inversión, sus características y dimensiones lo hacen ideal para la aplicación que se plantea (ver Figura 2.2). Además, el laboratorio de la Universidad está dotado de dos brazos de este tipo, por lo que se podrá realizar una prueba a escala real del proyecto.

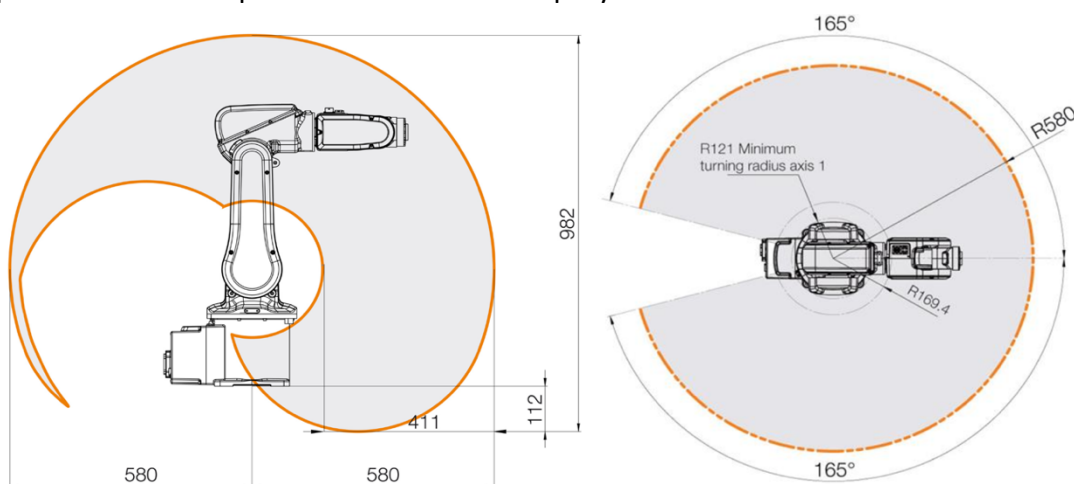


Figura 2.2. Área de trabajo del centro de la muñeca (eje 5) del robot IRB 120 (medidas en mm)

## 2.2. Controlador IRC5.

El controlador IRC5 está diseñado por la misma empresa que el brazo robótico (ABB). Este controlador es necesario para tener un manejo y un control total del robot IRB 120; es decir, es un equipo de control a través del cual se puede realizar la interacción máquina-humano. Este equipo basa su funcionamiento en el envío y recibo de señales utilizadas para controlar el brazo robótico. Además, posee una memoria donde se almacenan los diferentes programas que dirigen las operaciones que se deben realizar.

El controlador IRC5 viene equipado con un aparato denominado FlexPendant, que es un mando de control con una interfaz intuitiva que sirve para la programación y el manejo del robot (ver Figura 2.3). Consiste en un mando o joystick, con una pantalla táctil y distintos botones con los que poder programar y ejecutar los programas, configurar, e incluso monitorizar el estado del robot.



Figura 2.3. Controlador IRC5 y FlexPendant

## 2.3. RobotStudio.

El programa *RobotStudio* es un software de ingeniería, diseñado y patentado por la empresa ABB, y empleado para configurar y programar robots industriales, tanto robots físicos en el centro de producción como robots virtuales en un ordenador. Asimismo, permite un abanico de posibilidades en el mundo de la automatización industrial y la robótica manipuladora gracias a la versatilidad de su entorno de simulación.

RobotStudio se basa en la utilización de controladores virtuales de ABB, que son copias simuladas del software real que utilizan los robots de ABB. Gracias a esto, se puede ejecutar en el ordenador un sistema robótico que haya sido diseñado previamente antes de ser ejecutado en el robot real, con lo que se evitan costes innecesarios y reducción de riesgos.

Para la creación de la estación en 3D, el programa dispone de un amplio catálogo de robots, equipos y objetos. A pesar de esto, si se desea, el programa permite diseñar nuevos objetos o importarlos desde otros programas de modelado como *AutoCAD*, *SolidWorks*, *Solid Edge* o *CATIA*.

En este proyecto se ha utilizado la versión *RobotStudio 2019.2* y *RobotWare 6.10*.

Aunque este tema se abordará con mayor profundidad en el capítulo siguiente, en la simulación de la aplicación se dispondrá de dos robots IRB 120 con sus correspondientes herramientas y controladores, una camilla de intervención quirúrgica y un cuerpo humano, como se puede observar en la siguiente figura.

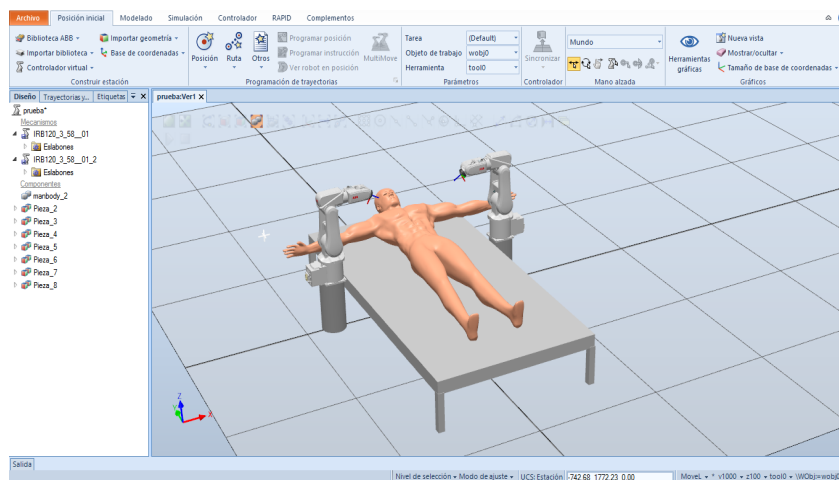


Figura 2.4. Interfaz de RobotStudio

## 2.4. Lenguaje de programación RAPID.

El software *RobotStudio* utiliza RAPID (Robotics Application Programming Interactive Dialogue), que es un lenguaje de programación textual de alto nivel desarrollado por la empresa ABB [19]. Una aplicación RAPID consta de un programa y una serie de módulos del sistema. A su vez, el programa es una secuencia de instrucciones que controlan el robot, de forma que realice las operaciones deseadas por el usuario final, y en general consta de tres partes (ver Figura 2.5):

- *Una rutina principal (main)*: rutina donde se inicia la ejecución.
- *Un conjunto de sub-rutinas*: sirven para dividir el programa en partes más pequeñas a fin de obtener un programa modular.
- *Los datos del programa*: definen posiciones, valores numéricos, sistemas de coordenadas, etc.

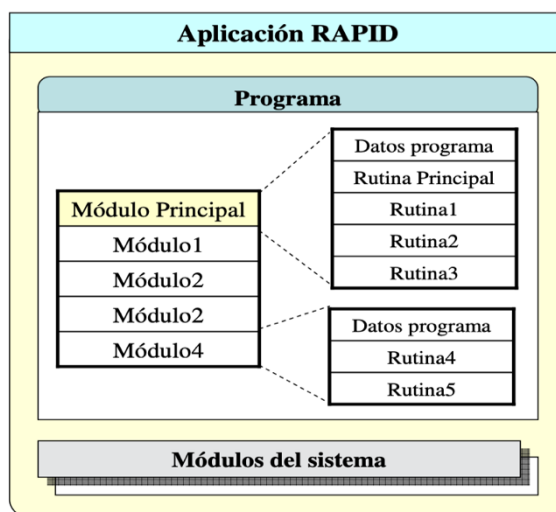


Figura 2.5. Estructura de aplicación RAPID



Este tipo de lenguaje es utilizado en este proyecto para conseguir que el robot realice los movimientos deseados que son recibidos en forma de datos mediante el protocolo de comunicación TCP/IP, cuyo funcionamiento se explicará en posteriores capítulos.

## 2.5. MATLAB.

*MATLAB* es un programa computacional que ejecuta una gran variedad de operaciones y tareas matemáticas. Es una herramienta potente y puede manejar los cálculos involucrados en problemas de ingeniería y ciencia. *MATLAB* integra el cálculo, la visualización y la programación en un ambiente fácil de utilizar donde los problemas y las soluciones se expresan en una notación matemática (ver Figura 2.6).

La ventaja principal que posee este programa es el uso de familias de comandos de áreas específicas llamadas *toolboxes*, que son grupos de comandos que extienden el ambiente de *MATLAB* para resolver problemas de áreas específicas de la ciencia e ingeniería. Por ejemplo, existen *toolboxes* para las áreas de Sistemas de Control, Redes Neuronales, Procesamiento Digital de Señales, etc.

Hay que destacar dos herramientas adicionales con las que *MATLAB* está equipado y que expanden sus prestaciones: *Simulink* y *GUIDE*, que son una plataforma de simulación multidominio y un editor de interfaces de usuario GUI respectivamente.

El cometido de este programa para el desarrollo del proyecto será la creación de una interfaz gráfica GUI que permita al usuario el control remoto del proceso y el movimiento del robot desde una pantalla intuitiva sin necesidad de modificar el código del programa. Esta interfaz gráfica GUI se ha desarrollado con la versión *MATLAB R2017b*.

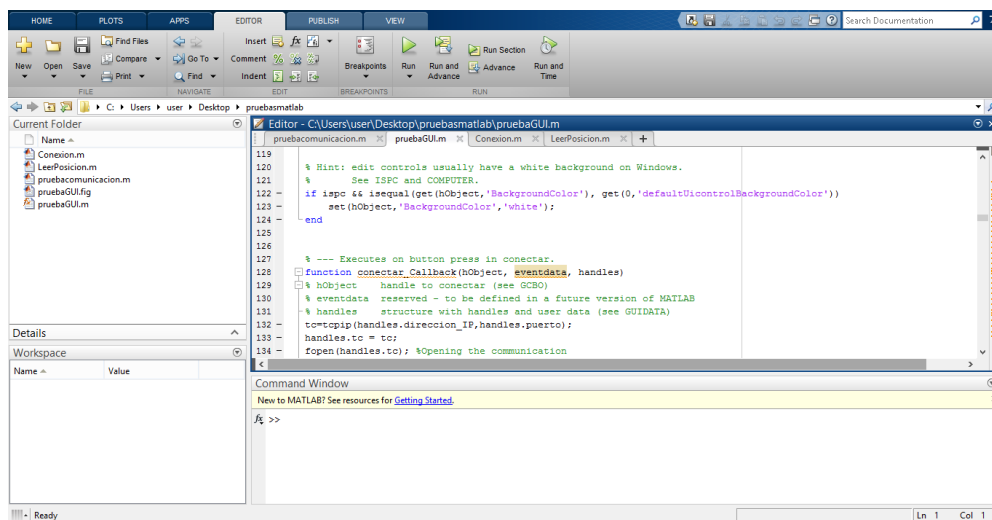


Figura 2.6. Interfaz de MATLAB

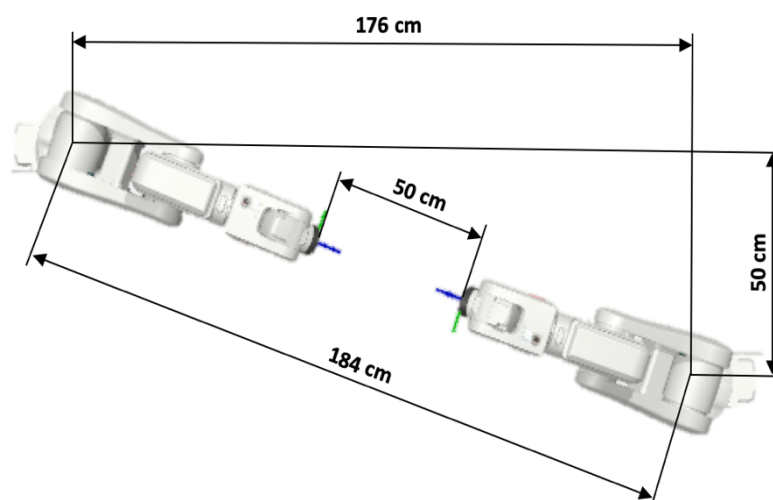
## CAPÍTULO 3. DISEÑO Y CONFIGURACIÓN DE LA ESTACIÓN DE TRABAJO EN ROBOTSTUDIO.

Este capítulo se centra en la explicación del diseño y la configuración de los distintos elementos de la estación de trabajo, de forma que tras su correspondiente programación funcionen de manera satisfactoria.

### 3.1. Diseño general de la estación.

Como se ha mencionado anteriormente, los elementos básicos de la estación de trabajo son los dos brazos robóticos IRB 120, con sus controladores y herramientas correspondientes, una camilla de operación y un cuerpo humano. Además, como se pretende que el trabajo desarrollado pueda ser evaluable en la realidad y se dispone de los medios necesarios en el laboratorio de la Universidad, se ha decidido replicar la estación de trabajo del laboratorio en el programa *RobotStudio*.

En la Figura 3.1 se muestra el plano en planta correspondiente a la estación, donde se han resaltado las medidas de mayor relevancia.



*Figura 3.1. Plano de la estación de trabajo*

Hay que tener en cuenta que debido a que la camilla de operación tiene una altura elevada y los brazos robóticos son de dimensiones considerablemente pequeñas, se necesitan crear unos soportes para cada uno de ellos con el fin de que se encuentren a una altura similar que la camilla y poder realizar los movimientos deseados correctamente.

Como se puede observar en dicho plano, la **mínima** distancia que separa a las muñecas de dichos robots cuando se encuentran directamente enfrentados es de 50 cm. Éste hecho, supone una importante limitación, pues no existe ningún punto en común que ambos robots puedan alcanzar para poder realizar una incisión y posterior sutura en el cuerpo humano.



A fin de solventar este problema, se ha decidido añadir un mango de larga longitud a la herramienta deseada y anclarlo a la muñeca de los robots para que ambos puedan operar en todo el torso humano.

### 3.2. Configuración de los controladores virtuales.

Para poder controlar los robots en el programa es necesario configurar un controlador virtual para cada uno de ellos. Más adelante, en estos controladores se cargarán los programas para realizar los movimientos oportunos que se vayan a dar durante la operación.

El primer paso a realizar es la importación de un robot a la estación. Esto se realiza en la pestaña '*Posición inicial*' → *Biblioteca ABB*, donde se selecciona el robot deseado, en este caso el robot IRB 120.

Una vez esté el robot en la estación, en la misma pestaña se debe seleccionar la opción *Controlador virtual* → *Desde diseño*. Para poder configurarlo de esta manera es necesario tener instalado algún paquete de *RobotWare*, que es un paquete del programa en el cual están instalados los controladores virtuales.

Una vez en la última pantalla, hay que presionar el botón de *Opciones* y seleccionar la categoría *Comunicación*. En ella, para poder controlar al robot desde una interfaz gráfica desde el ordenador es necesario activar la casilla **PC Interfaz** (ver Figura 3.2). Además, si se desea, se puede activar la casilla *FlexPendant Interfaz* para poder controlar el robot a través de este dispositivo.

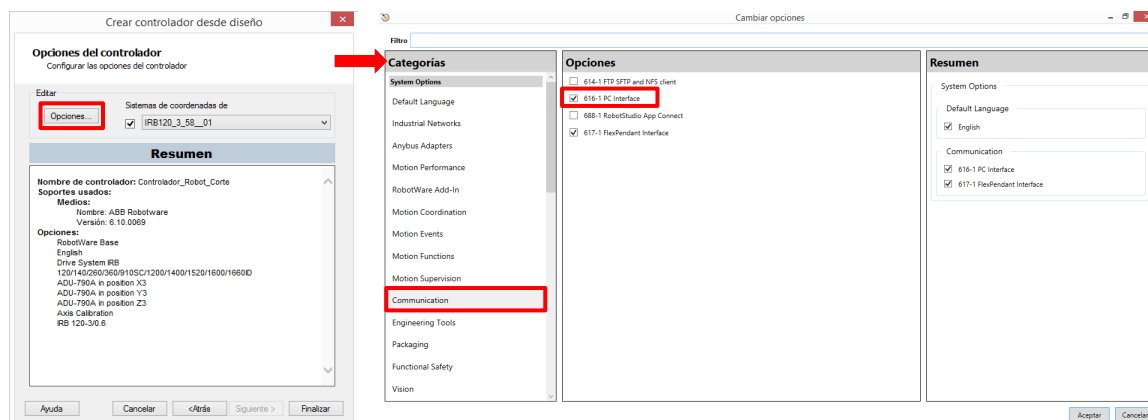


Figura 3.2. Configuración del controlador virtual

Como la estación de trabajo consta de dos brazos robóticos, se realiza el mismo procedimiento para la configuración del controlador virtual del segundo robot.

### 3.3. Diseño de las herramientas de trabajo.

La herramienta que se va a utilizar en este proyecto fue diseñada por el Departamento de Informática e Ingeniería de Sistemas de la Universidad y, tanto los planos como los archivos CAD fueron facilitados por el director de este Trabajo Fin de Grado para poder importarla a *RobotStudio*.

Se ha decidido utilizar esta herramienta debido a que ya está fabricada y a su multifuncionalidad, pues consta de dos partes muy convenientes para la aplicación de estudio. Una de estas partes es un **punzón** que simula un bisturí con el que realizar la incisión, mientras que la otra es una **pinza** que simula una grapadora médica con la que realizar la sutura. Es importante destacar que la pinza tiene dos posiciones, una abierta y otra cerrada, con unas aperturas de 7 cm y 1 cm respectivamente.

Gracias al programa *KeyShot* se ha podido efectuar un renderizado de la herramienta en el que se pueden observar estas dos partes mencionadas (ver Figura 3.3).

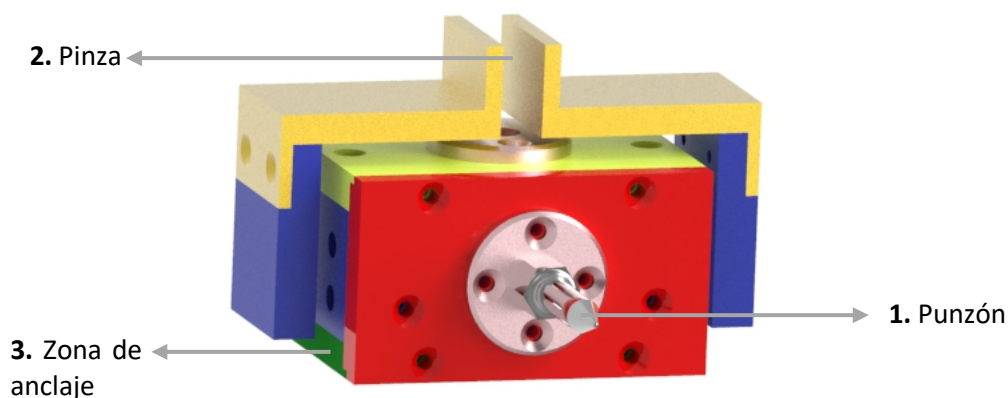


Figura 3.3. Renderizado de la herramienta

Debido al problema mencionado en el apartado 3.1. (falta de existencia de puntos comunes entre los robots), es necesario diseñar unos mangos para las herramientas para que éstas puedan alcanzar puntos comunes en el espacio. Además, como cada uno de los robots va a utilizar solo una de las dos partes de la herramienta, es necesario diseñar dos mangos diferentes para que cada parte de la herramienta esté correctamente orientada según su uso.

La zona de anclaje de la herramienta con la muñeca del robot se encuentra en la cara opuesta a la pinza (posición 3 en Figura 3.3). Además, se desea que la herramienta esté orientada hacia el suelo y en el punto medio de la pinza o en el punto central del punzón, dependiendo de cual se vaya a utilizar. Por dicho motivo, con ayuda del software *Solid Edge* se van a diseñar dos mangos diferentes, un mango recto y un mango con forma de L.

- **Herramienta encargada de realizar la incisión.** Como se ha mencionado anteriormente, para realizar la incisión se va a hacer uso del punzón, y como su eje debe ir en la dirección hacia el suelo, el mango diseñado para esta herramienta es recto.

- **Herramienta encargada de realizar la sutura.** En este caso, la pinza va a ser la encargada de suturar, y para conseguir la orientación adecuada se ha diseñado un mango con forma de L. Cabe recalcar que cuando esta herramienta alcance la posición en la que se debe suturar, la pinza se cerrará, simulando el proceso de grapar. Además, como la incisión y consecuente sutura pueden darse según diversas direcciones aleatorias, el mango posee un casquillo con retención axial que permite el giro de 360º de la herramienta.

Finalmente, en la siguiente figura se muestran las dos herramientas con sus mangos correspondientes.

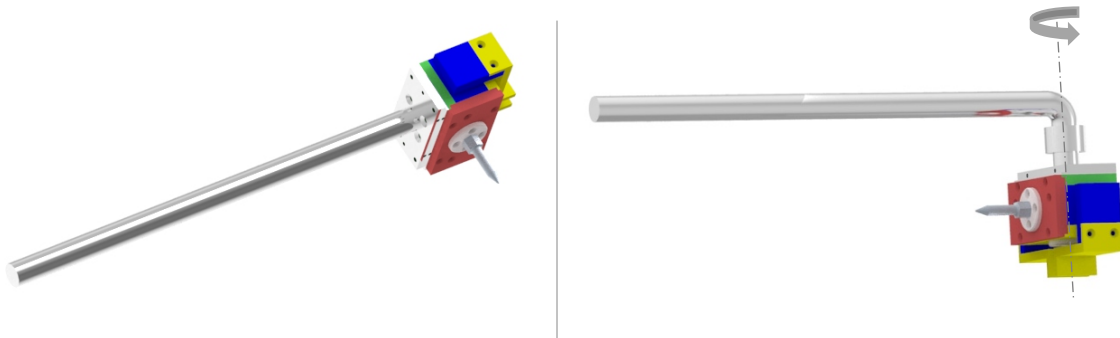


Figura 3.4. Herramientas de incisión y de sutura

En el Anexo I se pueden encontrar los planos con las dimensiones de estas herramientas.

### 3.4. Configuración de las herramientas de trabajo.

A continuación, se van a anclar las herramientas a las muñecas de cada uno de los robots. Para ello, en *RobotStudio* se va a importar la geometría de herramientas desde la pestaña de '*Posición inicial*' → *Importar geometría*. Como dicho programa admite la importación de varios tipos de archivos, se ha elegido utilizar el formato \*.obj ya que respeta los formatos de color, haciendo, por tanto, más visible la simulación.

Previo a esto, se necesita escalar la herramienta. Esto se debe a que, aunque los archivos CAD estuvieran en las unidades correctas, al guardarlo en formato \*.obj e importarlo a *RobotStudio*, la herramienta se importaba a escala 1000:1, es decir, el programa interpretaba las medidas en milímetros como metros. Por lo tanto, para conseguir las medidas deseadas en *RobotStudio* (escala 1:1) se debe escalar primero a 1:1000.

El procedimiento de escalado de una pieza en *Solid Edge* consta de los siguientes pasos (ver Figura 3.5):

1. Abrir un nuevo archivo en la plantilla **ISO métrico Pieza** para que el resultado obtenido sea una sola pieza en vez de un ensamblaje.
2. En la pestaña '*Inicio*' → *Portapapeles* → *Insertar copia de pieza* → Se selecciona la pieza que se desee escalar, en este caso, las herramientas.
3. Aparecerá una nueva ventana denominada **Parámetros de copia de pieza**, y en ella se escribe la escala que se desea, en este caso, una escala uniforme 0,001.
4. Finalmente, se exporta el archivo a formato \*.obj, listo para importar a *RobotStudio*.

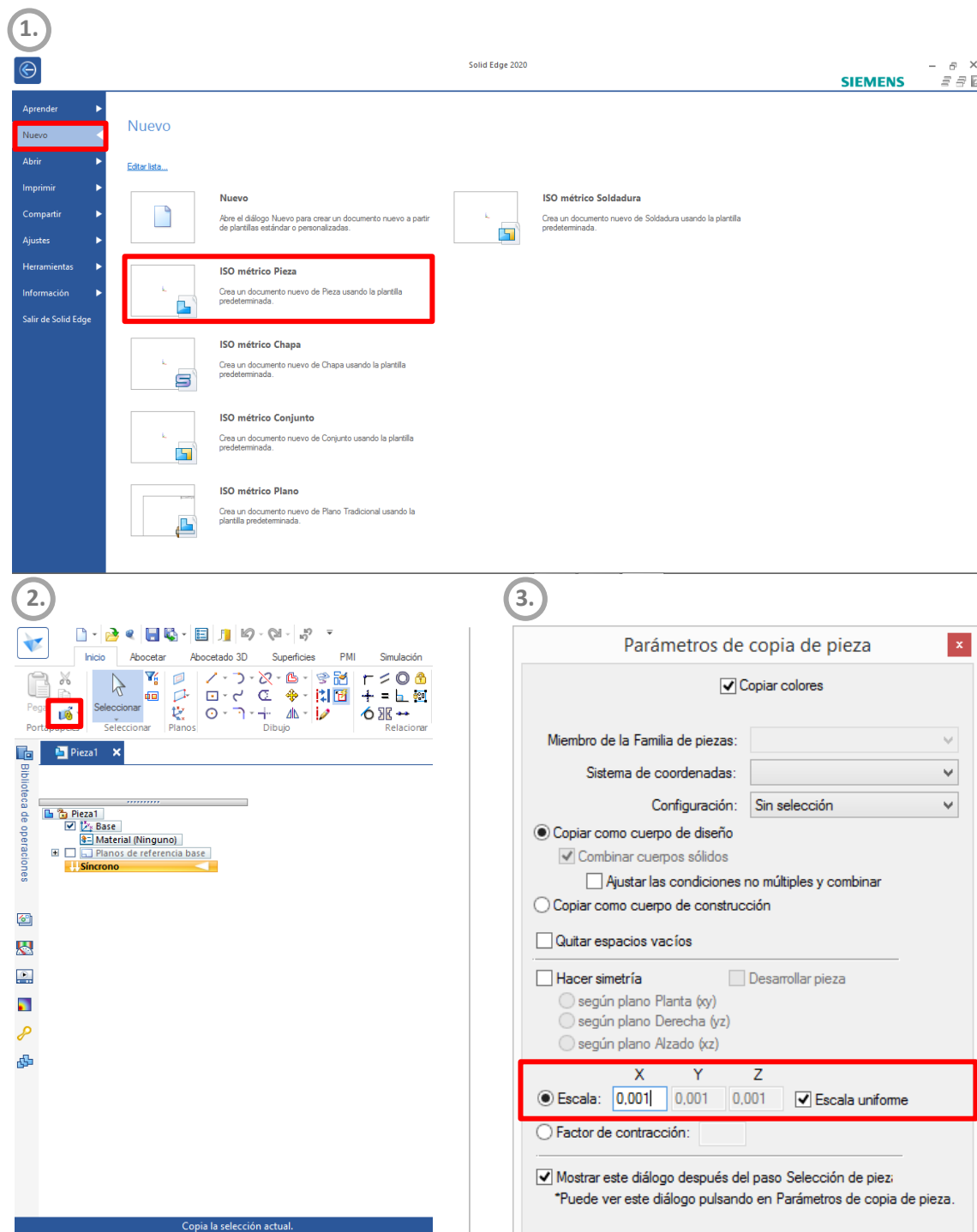


Figura 3.5. Proceso de escalado de las herramientas

Una vez importada, la herramienta aparece como una pieza (flecha verde en Figura 3.6) y no como una herramienta. Para poder convertir esta pieza en una herramienta hay que ir a la pestaña 'Modelado' → Mecanismo → Crear herramienta.

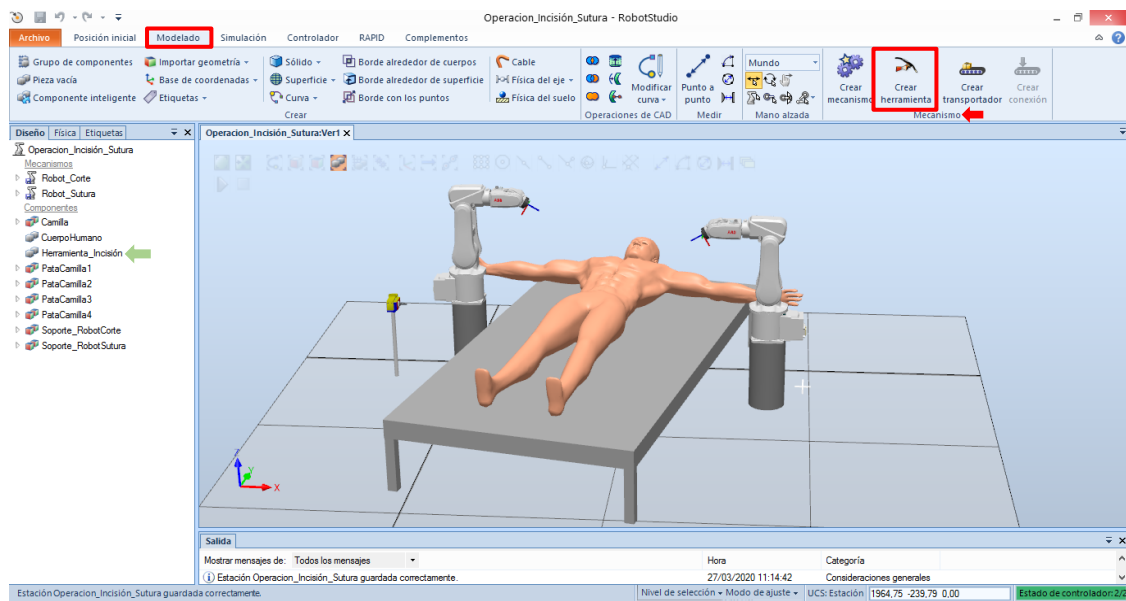


Figura 3.6. Creación de herramienta

Para la creación de la herramienta hay que seguir dos pasos. El **primero** de ellos es elegir la pieza que vamos a convertir en herramienta, en este caso, la pieza importada anteriormente, mientras que el **segundo** es posicionar el TCP de la herramienta (ver Figura 3.7). El **TCP** es el punto central de la herramienta que determina el punto de trabajo exacto de la herramienta y es el punto utilizado para el posicionamiento de los robots.

#### 1. Información de herramientas (paso 1 de 2)

Introduzca un nombre y seleccione el componente que está asociado a la herramienta.

Nombre:  
Herramienta\_Incisión

Seleccione componente:  
☒ Usar existente    ☐ Usar pieza simulada  
 Herramienta\_Incisión

Masa (Kg)  
1.00

Centro de gravedad (mm)  
 0.00 0.00 1.00

Momento de inercia  $I_x$ ,  $I_y$ ,  $I_z$  (kgm<sup>2</sup>)  
 0.00 0.00 0.00

Ayuda    Cancelar    < Atrás    Siguiente >

#### 2. Información de TCPs (paso 2 de 2)

Posicione sus TCPs y asígneles nombres.

Nombre de TCP:  
Herramienta\_Incisión\_TCP

TCP(s):  
Herramienta\_Incisión\_TCP

Valores del punto/sistema de

Posición (mm)  
 80.00 0.00 406.50

Orientación (deg)  
 0.00 0.00 0.00

Eliminar    Editar

Ayuda    Cancelar    < Atrás    Terminado

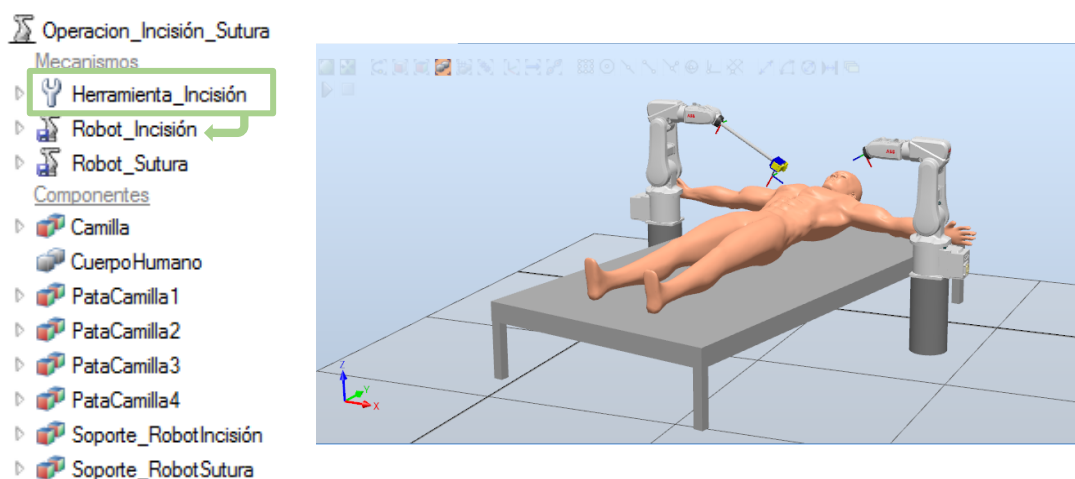
Figura 3.7. Pasos para la creación de la herramienta

En el primer paso se pueden asignar datos más técnicos a las herramientas, como su masa, centro de gravedad y momento de inercia en caso de ser datos conocidos, pero como en este caso no lo son, se dejan los valores predefinidos.

El segundo paso es el más importante, pues el TCP que se asigne será el punto de posición del robot. La posición del TCP de la herramienta hay que calcularla respecto a la muñeca del robot sobre el que se va a anclar la herramienta. Con ayuda de los ejes de la muñeca del robot y los planos del Anexo I donde aparecen las dimensiones de la herramienta, se determina la posición en la que el TCP ha de ser reposicionado de la siguiente manera:

- Como la herramienta es simétrica respecto del eje perpendicular al eje del punzón y, tanto la pinza como el punzón están centrados, la **coordenada Y** del TCP no sufre ningún cambio.
- El eje Z de la muñeca del robot es perpendicular a su superficie y, por lo tanto, el TCP de la herramienta estará desplazado, en la **coordenada Z**, tantos milímetros como longitud tenga el mango de la herramienta. En el caso de la herramienta de incisión esta medida es de 406,5 mm, mientras que en el de la herramienta de sutura es de 400 mm.
- Por otro lado, para calcular el desplazamiento del TCP en la **coordenada X**, se debe medir la distancia entre el eje del mango de la herramienta y el extremo del punzón o de la pinza, según cual se vaya a utilizar. Si hablamos de la herramienta de incisión, esta medida es de 80 mm, pero si hablamos de la herramienta de sutura es de 130 mm.

Finalmente, y como se puede observar en la Figura 3.8, la herramienta se ha creado correctamente. Para fijarla a la muñeca del robot lo único que hay que hacer es arrastrarla hasta el robot deseado.



*Figura 3.8. Fijación de la herramienta en el robot*

### 3.5. Configuración de funcionalidad de la herramienta de sutura.

El procedimiento a seguir para configurar la herramienta de sutura es diferente al utilizado para configurar la herramienta de incisión. Esto se debe a que, a la vez que se crea la herramienta, se le debe dar la **funcionalidad** necesaria, es decir, proporcionar el movimiento de rotación de 360° y el movimiento prismático para abrir y cerrar la pinza.

Primero, se debe importar la herramienta dividida en 4 partes diferentes: el mango, la herramienta central y los dos componentes de la pinza (ver Figura 3.9). Para conseguirlo, se debe seguir el procedimiento explicado en el apartado 3.4.

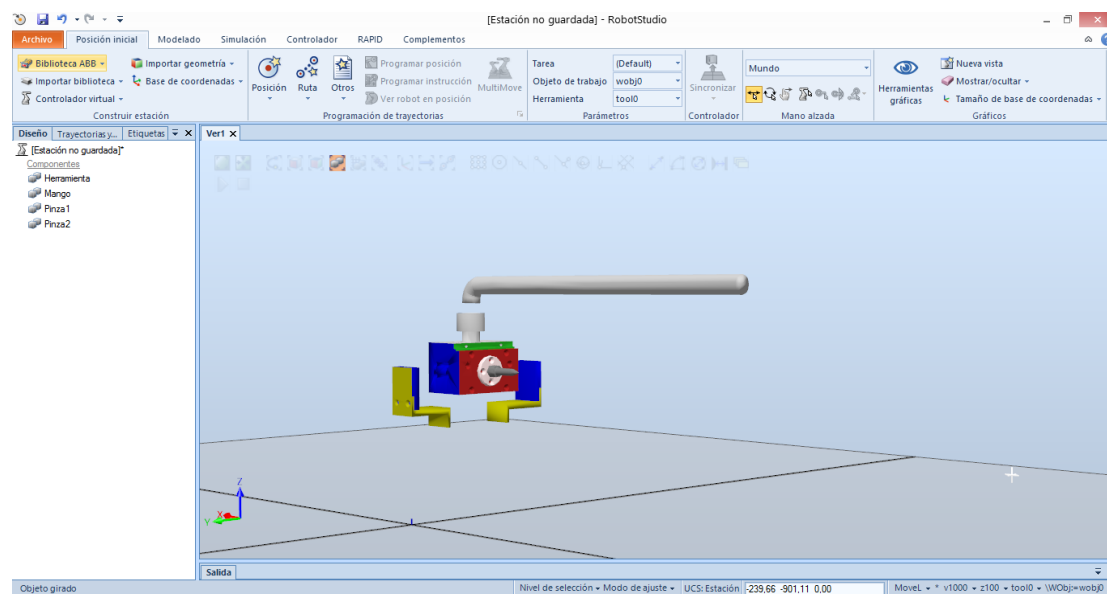


Figura 3.9. Importación herramienta separada en diferentes partes

A continuación, se mueven las diferentes partes de forma que éstas queden en la posición original que tiene la herramienta. Para que los dos componentes de la pinza realicen el movimiento de rotación simultáneamente con la herramienta central, es imprescindible **conectarlas** entre sí.

Después, para darle funcionalidad a la herramienta, hay que ir a la pestaña 'Modelado' → *Mecanismo* → *Crear mecanismo*.

Una vez pulsado *Crear mecanismo*, aparecerá una nueva ventana en la que por un lado hay que seleccionar el tipo de mecanismo que se va a crear (en este caso, herramienta) y por otro, definir los eslabones, ejes (articulaciones) y datos de la herramienta (ver Figura 3.10).

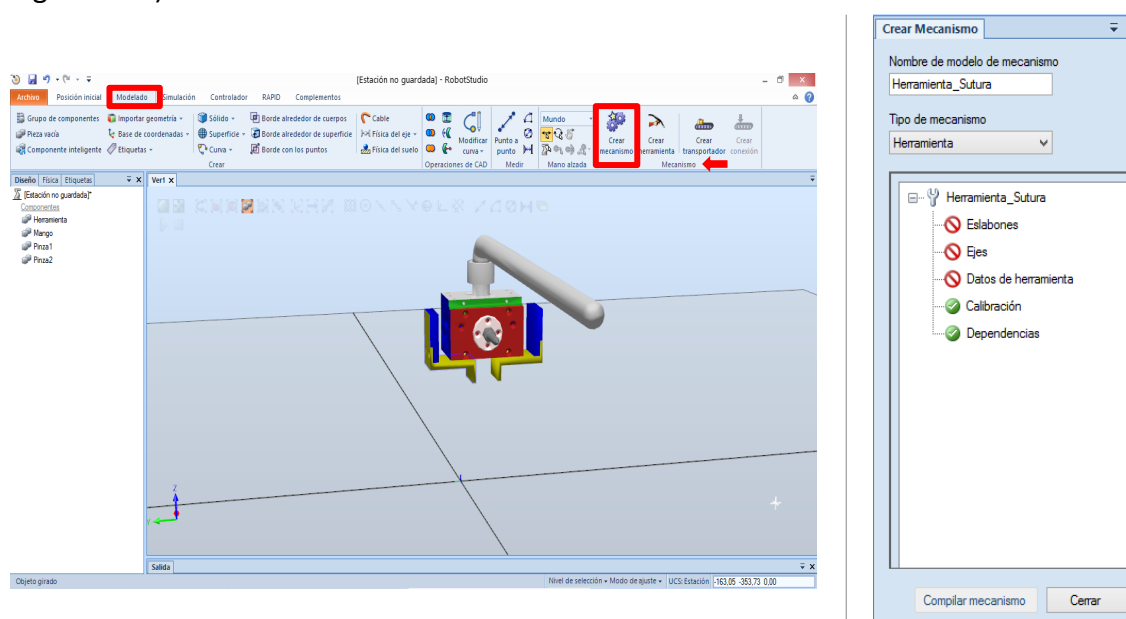


Figura 3.10. Creación de mecanismo para funcionalidad de herramienta



En lo que respecta a los eslabones y articulaciones, por definición, un manipulador robótico consta de una secuencia de elementos estructurales rígidos, denominados eslabones, conectados entre sí mediante juntas o articulaciones, que permiten el movimiento relativo de cada dos eslabones consecutivos. Por lo tanto, en este caso, existen **4 eslabones** (cada una de las 4 partes de la herramienta) y **3 articulaciones** (rotación 360º y movimiento prismático de cada componente de la pinza).

Para definir las articulaciones, se definen los siguientes parámetros:

1. El *tipo de articulación* (de rotación o prismática).
2. El *eslabón sobre el que se realiza el movimiento* y el *eslabón que lo realiza*.
3. El *eje* sobre el que se produce la rotación o el movimiento prismático.
4. Los *valores máximos y mínimos* de estos movimientos.

Figura 3.11. Creación de eslabones y articulaciones

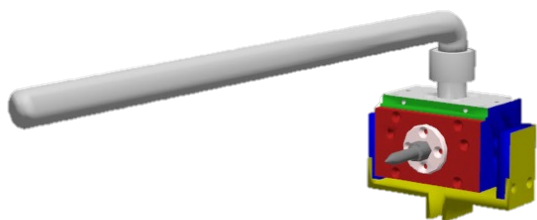
Para terminar de crear el mecanismo, se deben introducir los valores de la posición del TCP de la herramienta, [130,0,400], así como la masa, el centro de gravedad y los momentos de inercia de la misma en caso de ser conocidos. Tras esto, se presiona el botón *Compilar mecanismo*.

Una vez compilado, se pueden empezar a crear diferentes posiciones de la herramienta. En este caso, se han creado **8 posiciones** diferentes (ver Figura 3.12) para poder abarcar las direcciones de corte y poder suturar en esa dirección.

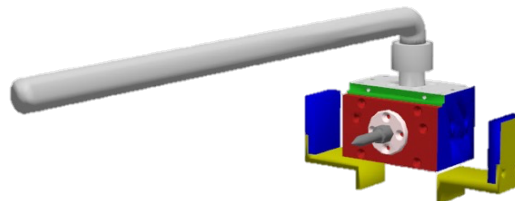
Se han definido 4 posiciones en diferentes direcciones, cada una de ellas girada unos grados diferentes respecto al eje de giro y en cada una se han definido dos posiciones de abertura de pinza: 7 cm (abierta) y 1 cm (cerrada). Esto se debe a que cuando la herramienta se encuentre en la posición en la que debe grapar, la pinza pasará de posición abierta (posición inicial) a posición cerrada, simulando la puesta de grapas.



1. Pinza normal cerrada 1 cm.



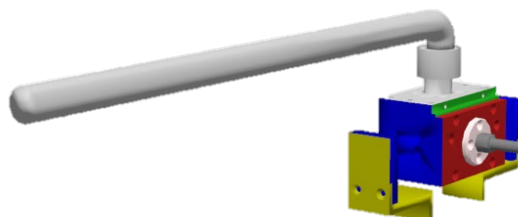
2. Pinza normal abierta 7 cm.



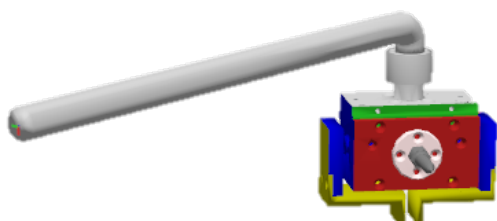
3. Pinza 90° cerrada 1 cm.



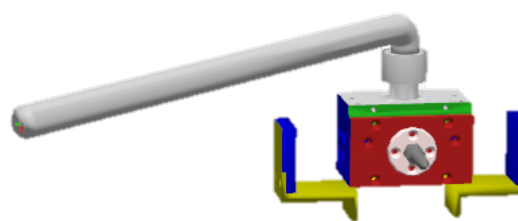
4. Pinza 90° abierta 7 cm.



5. Pinza 45° cerrada 1 cm.



6. Pinza 45° abierta 7 cm.



7. Pinza 135° cerrada 1 cm.



8. Pinza 135° abierta 7 cm.

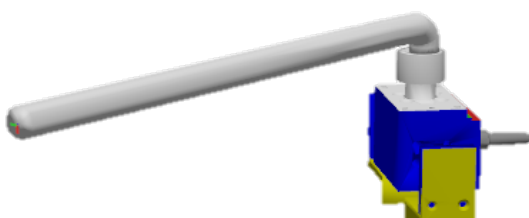


Figura 3.12. Posiciones de la herramienta

Además, si se desea, es posible definir tiempos de transición entre posiciones. En este caso, se han definido los siguientes tiempos de transición (ver Figura 3.13):

Definir tiempos de transición					
Tiempos de transición (s)					
A pose:	De pose:				
	Pose de sincronización	AbiertaNormal	CerradaNormal	Abierta90	Cerrada90
Pose de	0,000	0,000	0,000	0,000	0,000
AbiertaNormal	0,000	2,500	4,000	5,000	
CerradaNormal	0,000	2,500	5,000	4,000	
Abierta90	0,000	4,000	5,000	2,500	
Cerrada90	0,000	5,000	4,000	2,500	

Figura 3.13. Tiempos de transición entre posiciones

De esta manera ya se ha finalizado la creación del mecanismo de la herramienta y, para anclarla a la muñeca del robot hay que realizar lo explicado en el apartado 3.4.

Sin embargo, para que se pueda utilizar esta funcionalidad en la programación de la aplicación, se debe crear un **componente inteligente**, gracias al cual se puede diseñar la lógica que permite estos cambios de movimiento. La creación del mismo se da desde la pestaña 'Modelado' → *Componente Inteligente*.

Para este diseño, se hace uso de 4 componentes: dos *PoseMover*, un *LogicGate NOT* (puerta lógica) y un *LogicSRLatch* (Set-Reset). Además, se deben crear señales de entrada y de salida para que, activando y desactivando las mismas se consigan los movimientos de giro y apertura de la pinza.

Para ello se han creado **4 señales de entrada** (señales de acción: *AcciónNormal*, *Acción45*, *Acción90*, *Acción135*) para cada giro de la herramienta y **8 señales de salida** para cada una de las posiciones mostradas en la Figura 3.12. Cuando una de las acciones de giro se activa, la pinza pasa de posición abierta a cerrada o viceversa en la posición de giro en la que se encuentre la herramienta. Asimismo, cuando se desactiva la señal de acción, la pinza vuelve a la posición previa (abierta o cerrada) en la que se encontraba antes de la activación de dicha señal (ver Figura 3.14).

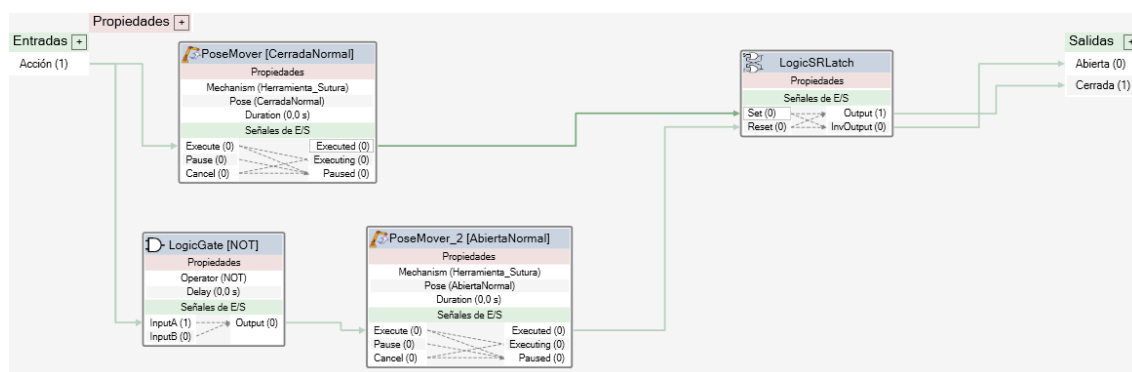


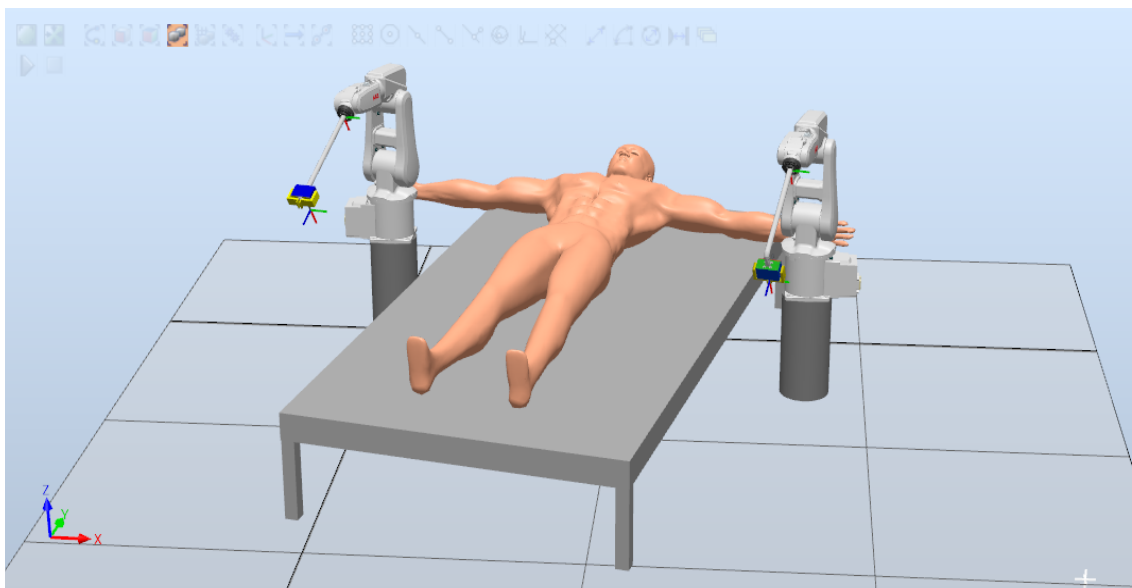
Figura 3.14. Lógica de la funcionalidad de la herramienta

Hay que destacar que la Figura 3.14 muestra únicamente la lógica de la funcionalidad de la pinza cuando ésta se encuentra en la posición neutral (giro 0º) donde la pinza se cierra cuando se activa la señal y se abre cuando se desactiva la misma.

En cambio, con las otras señales de acción ocurre lo contrario. Es decir, cuando la señal se activa, la herramienta pasa a la posición abierta girada los grados correspondientes a esa señal. A su vez, cuando la señal de acción se desactiva, la pinza se cierra manteniendo esa posición girada.

Tras lo visto, la herramienta se encuentra totalmente configurada para realizar su función y poder utilizarla en la programación de las aplicaciones de estudio.

Para concluir este capítulo, la Figura 3.15 muestra la estación de trabajo totalmente configurada y en la posición inicial en la que los robots se situarán cuando el programa se ejecute.



*Figura 3.15. Estación de trabajo final y posición inicial de los robots*

# CAPÍTULO 4. PROTOCOLO DE COMUNICACIÓN.

---

El modo más habitual de trabajar con un robot IRB 120 consistiría en crear un programa en lenguaje RAPID, simularlo en *RobotStudio*, cargarlo en el controlador IRC5 y finalmente ejecutarlo sobre el robot real.

Sin embargo, en este proyecto se plantea otra posibilidad de trabajo con el robot IRB 120, que consiste en realizar un control externo mediante un socket de comunicación. La palabra **socket** designa un concepto abstracto por el cual dos programas pueden intercambiar cualquier flujo de datos de manera fiable y ordenada, es decir, permiten una comunicación bidireccional. En este caso, esos dos programas son *RobotStudio* y *MATLAB*. Una de las ventajas más importantes de este tipo de conexión es que permite realizar aplicaciones complejas, como por ejemplo aquellas que hacen uso de una cámara de visión artificial.

Existe una amplia variedad de protocolos de comunicación (FTP, HTTP, TFTP, ARP, etc.), pero para realizar la comunicación del brazo robótico con el software matemático *MATLAB* en este proyecto, se va a utilizar el protocolo TCP/IP, puesto que su inmensa fiabilidad en la transmisión de datos asegura que no exista pérdida alguna de información.

Además, pensando en la aplicación real de este proyecto, la pérdida de información podría tener consecuencias devastadoras sobre la persona que esta siendo intervenida, por lo que el protocolo de comunicación seleccionado es el ideal.

## 4.1. Protocolo TCP/IP.

---

TCP/IP son las siglas de Protocolo de Control de Transmisión/Protocolo de Internet, que es un sistema de protocolos que hace posible la comunicación de ordenadores que no se encuentran en la misma red. El ejemplo más conocido de este protocolo es el servicio de email.

El **Protocolo de Control de Transmisión (TCP)** permite a dos anfitriones establecer una conexión e intercambiar datos. El TCP garantiza la entrega de datos, es decir, que los datos no se pierdan durante la transmisión; también garantiza que los paquetes sean entregados en el mismo orden en el cual fueron enviados.

El **Protocolo de Internet (IP)** utiliza direcciones que se pueden expresar mediante un número binario de 32 bits. Estos 32 bits se dividen en cuatro octetos, cuyo valor decimal está comprendido en el intervalo de 0 a 255, separados con un formato de punto decimal, por ejemplo, 192.168.254.6. Este protocolo es el encargado del enrutamiento, mecanismo por el que en una red los paquetes de información se hacen llegar desde su origen a su destino final, siguiendo un camino o ruta óptimo a través de la red.

Dependiendo de la extensión de la red, esto puede suponer transitar por muchos nodos intermedios.

La comunicación está compuesta por un servidor y un cliente (ver Figura 4.1):

- El **servidor** es programado en RAPID. El brazo robótico actúa como servidor y se ha programado para recibir la información enviada por el cliente en lo referido a posiciones, trayectorias, orientaciones, etc.
- El **cliente** es programado en *MATLAB*. Se encarga de procesar toda la información que el usuario introduce en la interfaz para enviarla correctamente al servidor.

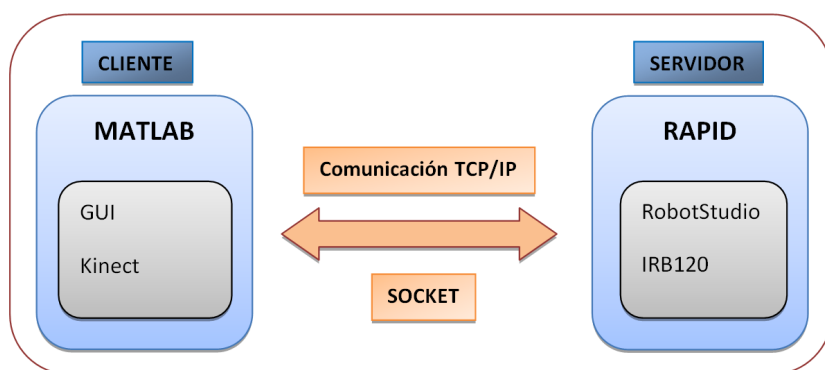


Figura 4.1. Estructura cliente-servidor

## 4.2. Establecimiento de la comunicación.

Para el establecimiento de la comunicación entre *MATLAB* y *RobotStudio* a través del protocolo TCP/IP, se necesita principalmente habilitar una dirección IP y un puerto al que se puedan conectar ambos sistemas. Cuando esto ocurra, se podrá empezar a transmitir información de un programa a otro.

Para que se produzca el intercambio de información oportuno, hay que crear unas líneas de código en cada uno de los programas para configurar en cada uno de ellos el servidor y el cliente, según corresponda.

Primero se establece la conexión en *RobotStudio*, es decir, se realiza la configuración del servidor. En el lenguaje de programación RAPID existen muchos tipos de variables, pero para establecer esta conexión nos interesan tres de ellas. La primera es una variable que se utiliza para la declaración del cliente y del servidor, denominada **socketdev**, mientras que las otras dos, **string** y **num**, se utilizan para declarar la dirección IP (en este caso, 127.0.0.1) y el puerto (en este caso, 41590) respectivamente.

```
VAR socketdev servidor;
VAR socketdev cliente;
VAR string direccion_IP:="127.0.0.1";
VAR num puerto:=41590;
```

Figura 4.2. Establecimiento de las variables en RobotStudio

Una vez declaradas las variables necesarias, se escribe el código principal que habilita la comunicación. Para ello se hará uso de una serie de comandos que se explican a continuación:

- **SocketCreate**: permite crear el nuevo socket.
- **SocketBind**: permite enlazar la comunicación indicándole el número de puerto y la dirección IP.
- **SocketListen**: permanece a la espera del cliente, actuando como servidor.
- **SocketAccept**: acepta la comunicación cuando el programa verifica que el cliente está conectado.

```
SocketCreate servidor;  
SocketBind servidor,direccion_IP,puerto;  
SocketListen servidor;  
SocketAccept servidor,cliente;
```

*Figura 4.3. Establecimiento de la conexión en RobotStudio*

Como en el caso de estudio intervienen dos brazos robóticos IRB 120, es decir, dos servidores, se deberá realizar lo mismo que se ha mencionado anteriormente. Por lo tanto, se tendrán dos variables de servidor, dos de dirección IP y dos de puerto, una de cada para cada uno de los robots, y una de cliente, que será la interfaz creada. Hay que tener en cuenta que estas líneas de código se cargan en el controlador de cada robot, por lo que habrá que hacer dos programas diferentes con el mismo código para establecer la conexión pero, cada uno con sus valores de dirección IP y puerto correspondiente.

A continuación, se establece esta comunicación en *MATLAB*. Como se ha comentado en el capítulo 2.5, este programa incorpora *toolboxes*. En particular, una de éstas proporciona una serie de comandos imprescindibles para trabajar con equipos externos, como son los dos robots IRB 120. El primer paso para configurar el cliente es crear un objeto **tcpip**, indicando la dirección IP y el puerto del servidor que se quiere controlar. Finalmente, para habilitar la comunicación, se utiliza el comando **fopen**.

```
tc = tcpip('127.0.0.1',41590); %Datos IP y puerto del servidor  
fopen(tc); %Habilitar la comunicación
```

*Figura 4.4. Establecimiento de la conexión en MATLAB*

De la misma forma que se ha procedido con anterioridad, se debe establecer la conexión con dos servidores, por lo que los comandos expuestos arriba se deben repetir, pero esta vez utilizando la dirección IP y el puerto del otro servidor.

Para realizar la simulación se han elegido unas direcciones IP y puertos arbitrarios, haciéndolos coincidir en ambos programas. En el momento en el que se realice la prueba en el laboratorio, cada uno de los robots tienen asignados una dirección IP y un puerto para conectarse. Por ejemplo, la dirección IP de uno de los robots es la siguiente: 10.3.17.206.

### 4.3. Transmisión de información.

Se puede pensar que, debido al diseño de una interfaz gráfica, los datos y valores recogidos en ella (cliente) se enviarán a los robots (servidor) y, por lo tanto, la comunicación sería unidireccional. Sin embargo, esto no es lo que sucede. Obviamente, la transmisión de información de *MATLAB* a *RobotStudio* es muy elevada por lo mencionado anteriormente, pero también se producirá la transmisión de datos de *RobotStudio* a *MATLAB*, consiguiendo la comunicación bidireccional. La mayor parte de esta transmisión será de comprobaciones para asegurarse de que el programa se ejecuta correctamente, aunque también habrá otros datos que enviar que se verán más adelante.

Como la emisión de datos se puede realizar desde cualquiera de los dos programas, dependiendo desde cual se efectúe se deben escribir unos comandos u otros que se comentan a continuación.

#### 4.3.1. Emisión de datos desde *MATLAB*.

En el caso de que la emisión se produzca desde *MATLAB*, el comando más importante es **fwrite** en el cual hay que introducir el socket de comunicación y el mensaje a enviar.

```
mensaje = 'Hola, esto es una prueba';  
fwrite(tc,mensaje); %Enviar el mensaje al servidor
```

*Figura 4.5. Ejemplo emisión de datos desde MATLAB*

#### 4.3.2. Recepción de datos desde *RobotStudio*.

Si el programa *RobotStudio* está a la espera de recibir información, el procedimiento utilizado será **SocketReceive**. En él se introduce como entrada: **1)** el tipo de mensaje que se espera recibir, es decir, si es string, byte o numero, **2)** el mensaje en cuestión y **3)** el tiempo máximo en el que se espera recibirlo.

```
SocketReceive cliente, \Str:=mensaje \Time:=5;  
!El mensaje recibido se guarda en la variable mensaje
```

*Figura 4.6. Ejemplo recepción de datos desde RobotStudio*

#### 4.3.3. Emisión de datos desde *RobotStudio*.

En este caso, se va a utilizar el procedimiento **SocketSend** que tiene una estructura similar a la de **SocketReceive** explicada anteriormente, pero sin la entrada de tiempo máximo.

Además, deberemos obtener la longitud del mensaje y enviarla también a *MATLAB* debido a que si no se puede producir un error bastante común que se comentará en el siguiente subapartado.

Si el mensaje que se desea transmitir es de tipo string, para obtener su longitud se utilizará la función **StrLen**. En cambio, si es un número, primero hay que transformar el valor en string con la función **ValToStr** y finalmente utilizar **StrLen**. Es importante destacar que esta función nos devuelve un valor numérico y, por lo tanto, si se desea enviar como string, que es lo más común, se deberá volver a transformar a string con la función mencionada con anterioridad.

```
mensaje:="Hola, esto es una prueba";
longitud:=ValToStr(StrLen(mensaje));
!Se obtiene el valor numérico de la longitud del mensaje y lo transforma a string para enviarlo
SocketSend cliente, \Str:=longitud;
SocketSend cliente, \Str:=mensaje;
```

*Figura 4.7. Ejemplo emisión de datos desde RobotStudio*

#### 4.3.4. Recepción de datos desde MATLAB.

Cuando **MATLAB** está esperando el recibo de un mensaje, el comando utilizado es **fread** en el cual hay que introducir también el socket de comunicación y la longitud que se desea leer. Esto último es realmente importante pues, de lo contrario, el programa no detectará cuando debe terminar de leer el mensaje, produciendo una pérdida de eficacia y resultando, casi inevitablemente, en un error.

La longitud que se desea leer puede ser introducida como la longitud máxima y, por lo tanto leerá todo el mensaje, o como un intervalo, como por ejemplo [1,5] y leerá únicamente de la posición uno a la cinco del mensaje.

```
longitud = str2double(char(fread(tc,2)));
%1. Leer la longitud del mensaje (lee 2 caracteres porque el mensaje no va a tener más
de 2 cifras, es decir, la longitud del mensaje no será mayor de 99 caracteres)
%2. Transformar a carácter para que str2double funcione
%3. Transformar de string (array of char) a double (num)
mensaje = fread(tc,[1,longitud]); %Leer desde el carácter 1 hasta longitud
```

*Figura 4.8. Ejemplo recepción de datos desde MATLAB*

#### 4.3.5. Transmisión de posiciones del robot.

Un objeto de estudio va a ser el caso de la transmisión de las posiciones del robot bidireccionalmente, ya que va a servir para poder implementarlo más adelante en la interfaz gráfica.

- Para enviar la posición actual en la que se encuentra el robot de *RobotStudio* a *MATLAB* es imprescindible utilizar la función **CPos**, que tiene como argumentos de entrada la herramienta y el espacio de trabajo, y como argumento de salida posición actual del TCP de la herramienta. Una vez conocidas las tres componentes de la posición, éstas y sus longitudes se envían al cliente como se ha hecho anteriormente.

Mientras tanto, el cliente lee cada uno de estos mensajes y los transforma de string a número para que puedan ser utilizadas de forma numérica en el programa.

A continuación se muestra el código que representaría esta situación.



```

posicion:=CPos(\Tool:=tool0 \WObj:=wobj0);
posicionX:=ValToStr(posicion.x);
posicionY:=ValToStr(posicion.y);
posicionZ:=ValToStr(posicion.z);
longitudX:=ValToStr(StrLen(posicionX));
longitudY:=ValToStr(StrLen(posicionY));
longitudZ:=ValToStr(StrLen(posicionZ));
SocketSend cliente, \Str:=longitudX;
SocketSend cliente, \Str:=longitudY;
SocketSend cliente, \Str:=longitudZ;
SocketSend cliente, \Str:=posicionX;
SocketSend cliente, \Str:=posicionY;
SocketSend cliente, \Str:=posicionZ;
-----
%Leer solo un carácter de longitud porque va a ser menor que 10
longitudX = str2double(char(fread(tc,1)));
longitudY = str2double(char(fread(tc,1)));
longitudZ = str2double(char(fread(tc,1)));
%Leer las coordenadas
coordenadaX = str2double(char(fread(tc,[1,longitudX])));
coordenadaY = str2double(char(fread(tc,[1,longitudY])));
coordenadaZ = str2double(char(fread(tc,[1,longitudZ])));
%Unir coordenadas

```

*Figura 4.9. Transmisión de posiciones de RobotStudio a MATLAB*

- En cambio, si lo que se desea es enviar una posición al robot desde *MATLAB* y que éste se mueva hacia ella, lo primero que hay que hacer es definir estas coordenadas y después estos valores transformarlos string (**num2double**), y de ahí coger su carácter (char). Esto es muy importante porque cuando se envíe la posición completa, lo que se va a enviar es un string, es decir, un conjunto de caracteres.

En el momento en el que se tengan las tres coordenadas en tipo char, se van a unir en un string con el formato: [x,y,z] y se enviarán al servidor.

Una vez el servidor reciba las coordenadas, las tendrá que procesar, es decir, dividir las en x, y, z por separado para poder entenderlas como números y moverse a esa posición. Para lograr esto, se utilizan varias funciones que se explican a continuación:

- **StrFind**: busca un carácter en un string.
- **StrLen**: obtiene la longitud de un string.
- **StrPart**: busca una parte de un string y lo renombra.

Después, se unen en el formato correspondiente a posición en *RobotStudio*, que es [x,y,z], y con el procedimiento **MoveL** o **MoveJ** se mueve el TCP de la herramienta del robot a esa posición de forma lineal o mediante la sucesiva combinación de movimientos en los diferentes ejes respectivamente.

Finalmente, el servidor enviará un mensaje de comprobación ('Ok') al cliente para indicar que lo ha recibido correctamente.

```

%Escribir coordenadas en mm
x = 200;
y = 40;
z = 300;
X = char(num2str(x));
Y = char(num2str(y));
Z = char(num2str(z));
%Unir coordenadas y enviar
posicion = [X, ',', Y, ',', Z];
fwrite(tc, posicion);
mensaje = fread(tc, 2);
waitfor(mensaje, 'Ok');

-----

SocketReceive cliente, \Str:=coordenadas \Time:=20;
Pos1:=StrFind(coordenadas,1,"");
Pos2:=StrFind(coordenadas,Pos1+1,"");
Pos3:=StrLen(coordenadas);
Ok:=StrToVal(StrPart(coordenadas,1,Pos1-1),x);
Ok:=StrToVal(StrPart(coordenadas,Pos1+1,Pos2-Pos1-1),y);
Ok:=StrToVal(StrPart(coordenadas,Pos2+1,Pos3-Pos2),z);
posicion:=[x,y,z];
MoveJ[posicion,[1,0,0,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09],v10,fine,tool0\WObj:=wobj0;
SocketSend cliente, \Str:=comprobacion;

```

*Figura 4.10. Transmisión de posiciones de MATLAB a RobotStudio*

## 4.4. Cierre de la conexión.

Como se ha mencionado anteriormente, existen dos servidores y, por tanto, hay que establecer conexión con ambos a la vez para realizar aplicaciones que necesiten el trabajo simultáneo de los robots. Además, en el momento que uno de los robots haya terminado su trabajo, se deberá cerrar la conexión entre el cliente y ese servidor en concreto. Esto es posible gracias al comando **fclose**.

Finalmente, el código de establecimiento y cierre de conexión quedaría de la manera expuesta en la siguiente figura.

```

tc_1 = tcpip('127.0.0.1',41590);
tc_2 = tcpip('127.0.0.11',41595);
fopen(tc_1);
fopen(tc_2);
%Código del primer robot
%Código del segundo robot
fclose(tc_1);
fclose(tc_2);

```

*Figura 4.11. Establecimiento y cierre de la conexión*

# CAPÍTULO 5. MODELADO CINEMÁTICO DEL ROBOT IRB 120.

Una vez establecida la conexión explicada en el capítulo anterior, se deben realizar todos los cálculos necesarios para controlar el movimiento de los brazos robóticos correctamente y que los programas conectados se envíen la información correspondiente a estos movimientos. Este capítulo se basa principalmente en el modelo geométrico directo, que sirve para que la articulación terminal del robot alcance la posición cartesiana deseada cambiando únicamente los ángulos de cada una de sus articulaciones. Cabe destacar que *RobotStudio* posee comandos que incluyen directamente el modelo geométrico directo y que este desarrollo manual no es necesario para la consecución de los objetivos. Sin embargo, tiene interés académico para profundizar en los conocimientos y comparar el desarrollo manual con el modelo de *RobotStudio*.

## 5.1. Modelo geométrico directo de posición: Fundamento teórico.

El modelo geométrico directo define la orientación y posición del extremo final del robot basándose en las variables que definen sus respectivas articulaciones. Como el robot de estudio es de configuración angular (brazo articulado), las variables que definen sus articulaciones son los ángulos girados por cada uno de sus ejes.

$$\begin{aligned} \mathbf{q} &= (q_1, q_2, q_3, q_4, q_5, q_6)^T \\ \mathbf{X} &= (x, y, z, \phi, \theta, \psi)^T \\ \mathbf{X} &= \mathbf{f}(\mathbf{q}) \end{aligned}$$

donde :

- $\mathbf{q}$  representa el conjunto de **coordenadas articulares** que definen posición del extremo del robot. Estas coordenadas articulares son 6 debido a que se corresponden con los grados de libertad del robot.
- $\mathbf{X}$  se corresponde con el término **localización** de esa misma posición. Este término aúna la posición  $(x, y, z)$  y la orientación  $(\phi, \theta, \psi)$  del elemento terminal del robot, sumando así 6 grados de libertad.

Para su representación se utilizan unas matrices denominadas **matrices de transformación homogénea** (ver Figura 5.1). Esta matriz expresa la posición y orientación de un sistema de coordenadas  $\{S_{i-1}\}$  respecto a otro  $\{S_i\}$ . Dicha matriz está compuesta por cuatro submatrices de distinto tamaño que representan la rotación, translación, perspectiva y escalado.

$$\mathbf{T} = \begin{bmatrix} \text{Rotación} & \text{Traslación} \\ \text{Perspectiva} & \text{Escalado} \end{bmatrix} = \begin{bmatrix} \mathbf{R}_{3 \times 3} & \mathbf{p}_{3 \times 1} \\ \mathbf{f}_{1 \times 3} & w_{1 \times 1} \end{bmatrix}$$

Figura 5.1. Matriz de transformación homogénea

- $R_{3 \times 3}$ : Matriz de rotación.
- $P_{3 \times 1}$ : Vector de traslación.
- $f_{1 \times 3}$ : Vector de perspectiva (visión artificial). Para robótica:  $f = (0,0,0)$ .
- $w_{1 \times 1}$ : Factor de escala. Si no se indica lo contrario, siempre se tomará  $w = 1$ .

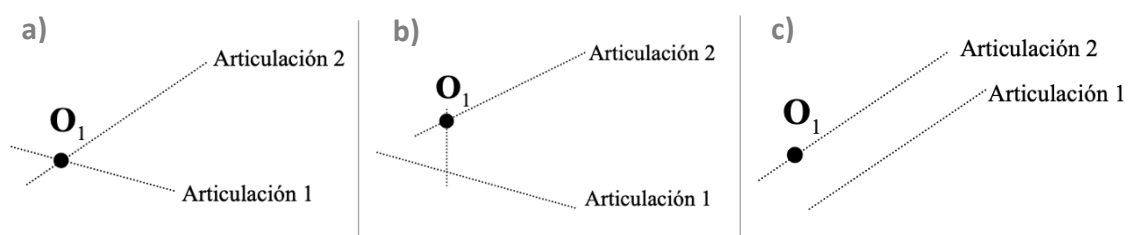
En robótica se hace uso de la representación **Denavit-Hartenberg** (D-H) para describir la relación que existe entre dos eslabones contiguos. Esta representación establece la localización que debe tomar cada sistema de coordenadas ligado a cada eslabón  $i$ -ésimo de una cadena articulada para poder sistematizar la obtención de las ecuaciones cinemáticas de la cadena completa [20].

La matriz de transformación homogénea queda definida por 6 grados de libertad (traslación y rotación). Sin embargo, el método Denavit-Hartenberg permite, siempre y cuando se elijan adecuadamente los correspondientes sistemas coordenados, reducir el número de grados de libertad a 4. Para esta correcta elección de los sistemas de coordenadas hay que seguir los siguientes pasos [21]:

Primero, se numeraran las articulaciones empezando en 1 y los eslabones empezando en 0. Por lo tanto, la base fija es el eslabón 0 y la articulación  $i$  une los eslabones  $i-1$ ,  $i$ .

Después, para cada eslabón  $i$ -ésimo se sitúa un sistema de referencia solidario (sistema de referencia asociado a cada eslabón) de la siguiente manera:

- Se coloca el origen  $O_i$  en el eje de la articulación  $i+1$  y, dependiendo de los ejes de sus articulaciones se coloca en una posición u otra acorde a este criterio:
  - Si los ejes de las articulaciones  $i$ ,  $i+1$  se **cortan**: se coloca  $O_i$  en el punto de intersección de ambos ejes. (ver Figura 5.2.a)
  - Si los ejes de las articulaciones  $i$ ,  $i+1$  se **cruzan**: se coloca  $O_i$  en el punto de intersección con la perpendicular común a ambos ejes. (ver Figura 5.2.b)
  - Si los ejes de las articulaciones  $i$ ,  $i+1$  son **paralelos**: se coloca  $O_i$  en cualquier punto del eje de la articulación  $i+1$ . (ver Figura 5.2.c)



*Figura 5.2. Situación del Origen de los sistemas de coordenadas según D-H*

- Se sitúa el vector  $Z_i$  sobre el eje de la articulación  $i+1$ .

- Se sitúa el vector  $X_i$  en la perpendicular común a los ejes de las articulaciones  $i$ ,  $i+1$  (paralelo o antiparalelo al producto vectorial:  $Z_{i-1} \times Z_i$ ).

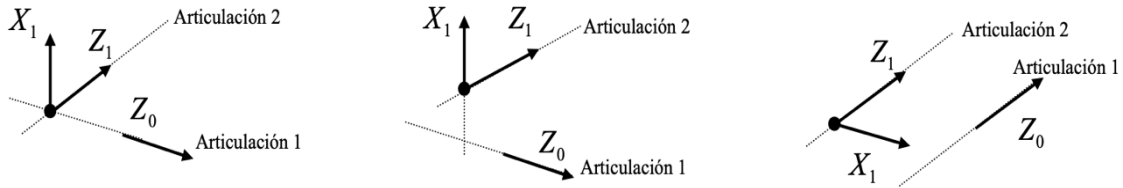


Figura 5.3. Situación del vector  $X$  de los sistemas de coordenadas según D-H

- Se sitúa el vector  $Y_i$  de forma que se cree un triedro directo  $Y_i = Z_i \times X_i$ .

Finalmente, hay una mayor libertad para el primer y el último sistema de coordenadas, (0) y (n-ésimo) respectivamente, ya que el vector  $Z_0$  se sitúa sobre el eje de la primera articulación, mientras que el vector  $X_n$  se coloca perpendicular a  $Z_{n-1}$ .

Escogiendo los sistemas de coordenadas asociados a cada eslabón según lo explicado, es posible pasar de un sistema de coordenadas al siguiente mediante cuatro transformaciones básicas.

Estas cuatro transformaciones básicas consisten en una sucesión de rotaciones y traslaciones que permiten relacionar el sistema de referencia del elemento  $i-1$  con el sistema del elemento  $i$ . Las transformaciones en cuestión son las siguientes:

1. Rotación alrededor del eje  $Z_{i-1}$  un ángulo  $\theta_i$ .
2. Traslación a lo largo de  $Z_{i-1}$  una distancia  $d_i$ .
3. Traslación a lo largo de  $X_i$  una distancia  $a_i$ .
4. Rotación alrededor del eje  $X_i$  un ángulo  $\alpha_i$ .

Debido a que el producto de matrices no es conmutativo, las transformaciones se han de realizar en el orden indicado. De este modo se tiene que la **matriz simbólica** tiene la siguiente forma:

$${}^{i-1}T_i = \text{Rot}(Z, \theta_i) \cdot \text{Trasl}(Z, d_i) \cdot \text{Trasl}(X, a_i) \cdot \text{Rot}(X, \alpha_i)$$

$${}^{i-1}T_i = \begin{bmatrix} \cos\theta_i & -\text{sen}\theta_i & 0 & 0 \\ \text{sen}\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\text{sen}\alpha_i & 0 \\ 0 & \text{sen}\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^{i-1}T_i = \begin{bmatrix} \cos\theta_i & -\text{sen}\theta_i \cdot \cos\alpha_i & \text{sen}\theta_i \cdot \text{sen}\alpha_i & a_i \cdot \cos\theta_i \\ \text{sen}\theta_i & \cos\theta_i \cdot \cos\alpha_i & -\cos\theta_i \cdot \text{sen}\alpha_i & a_i \cdot \text{sen}\theta_i \\ 0 & \text{sen}\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

donde  $a_i$ ,  $\alpha_i$ ,  $d_i$ ,  $\theta_i$  son los parámetros Denavit-Hatemberg del eslabón  $i$ -ésimo y se definen de la siguiente forma:

- $a_i$  : se corresponde con la longitud del eslabón  $i$  o la distancia entre ejes de las articulaciones  $i, i+1$ . Es la distancia entre el eje  $Z_{i-1}$  y  $Z_i$  a lo largo de  $X_i$ .
- $\alpha_i$  : es el ángulo de torsión del eslabón  $i$ . Es el ángulo de separación del eje  $Z_{i-1}$  y  $Z_i$  medido respecto al eje  $X_i$ .
- $d_i$  : es la distancia desde  $O_{i-1}$  hasta la intersección del eje  $Z_{i-1}$  con el eje  $X_i$  a lo largo de  $Z_{i-1}$ .
- $\theta_i$  : es el ángulo que forman los ejes  $X_{i-1}$  y  $X_i$  respecto del eje  $Z_{i-1}$ .

Estos cuatro parámetros ( $a, \alpha, d, \theta$ ) dependen únicamente de las características geométricas de cada eslabón y de las articulaciones que las unen con el anterior y el posterior (ver Figura 5.4).

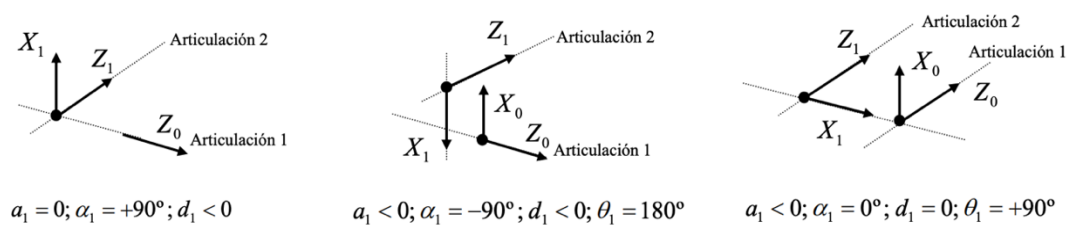


Figura 5.4. Parámetros Denavit-Hartenberg

Una vez obtenidos los parámetros D-H, el modelo geométrico directo está definido y el cálculo de las relaciones entre los eslabones consecutivos del robot es inmediato a partir de las matrices  ${}^{i-1}\mathbf{T}_i$ . El producto de multiplicar todas las matrices obtenidas da como resultado una matriz de transformación homogénea que expresa la orientación y posición del extremo del robot en función de sus coordenadas articulares.

## 5.2. Modelo geométrico directo de posición del robot IRB 120.

Para empezar el análisis del modelo geométrico directo del brazo robótico IRB 120 se debe estudiar la geometría del mismo [22] que se muestra en la Figura 5.5.

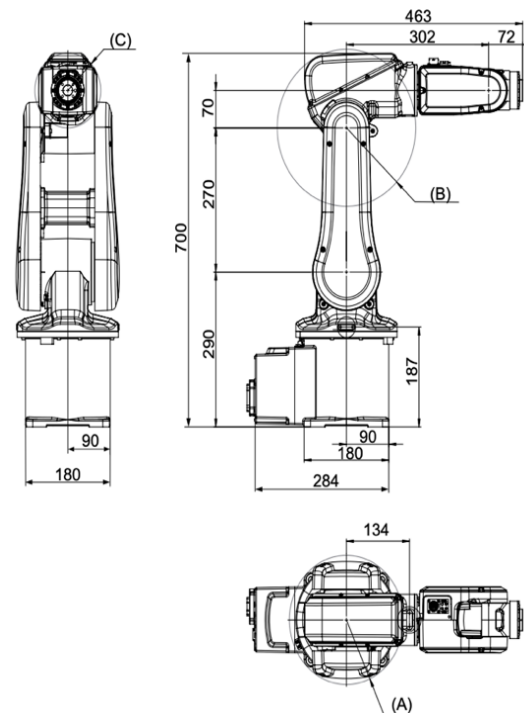


Figura 5.5. Dimensiones del robot IRB 120

Para continuar con el análisis, se deben identificar los eslabones y las articulaciones del robot y nombrarlas de acuerdo a las reglas del método Denavit-Hartenberg (ver Figura 5.6). Como se ha comentado anteriormente, el robot IRB 120 tiene 6 grados de libertad y todas sus articulaciones son de tipo rotatorio.

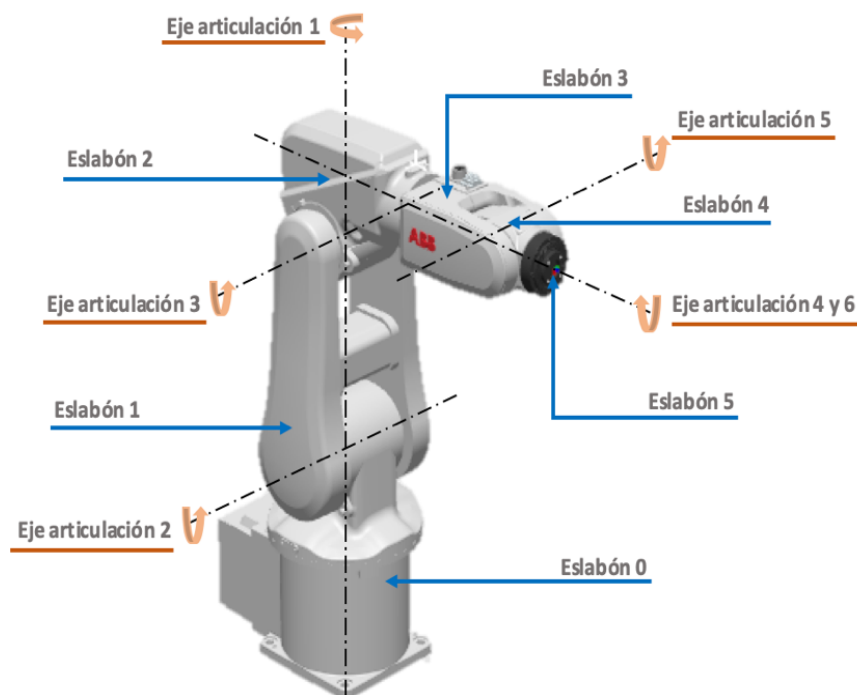


Figura 5.6. Eslabones y articulaciones del robot IRB 120

Con los eslabones y los ejes de las articulaciones ya identificados, se sitúan los sistemas de coordenadas asociados a cada eslabón del brazo robótico. Desde su posición inicial, se van colocando los sistemas coordenados desde la base hasta el extremo del robot, asignando el 0 a la base y el  $n-1$  a la última articulación, siendo  $n$  el número de grados de libertad o articulaciones. Es decir, el sistema de coordenadas inicial  $\{S_0\}$  se corresponde con el eslabón 0 y la articulación 1. Por último, el sistema de coordenadas  $n$  se ubicará en el extremo del robot (ver Figura 5.7).

Cabe destacar que el sistema de coordenadas de la última articulación  $\{S_5\}$  se ha situado desplazado para un mejor visionado debido a que el origen coincidía con el origen del sistema de coordenadas de la penúltima articulación  $\{S_4\}$ .

Además, a modo de aclaración, en cada sistema de coordenadas de la Figura 5.7 se ha dibujado un círculo en el eje sobre el que gira la articulación. Como se puede observar, el último sistema de coordenadas  $\{S_6\}$  no tiene dibujado el círculo pues dicho sistema coordenado no tiene asignada ninguna articulación, sino que únicamente sitúa el extremo del robot.

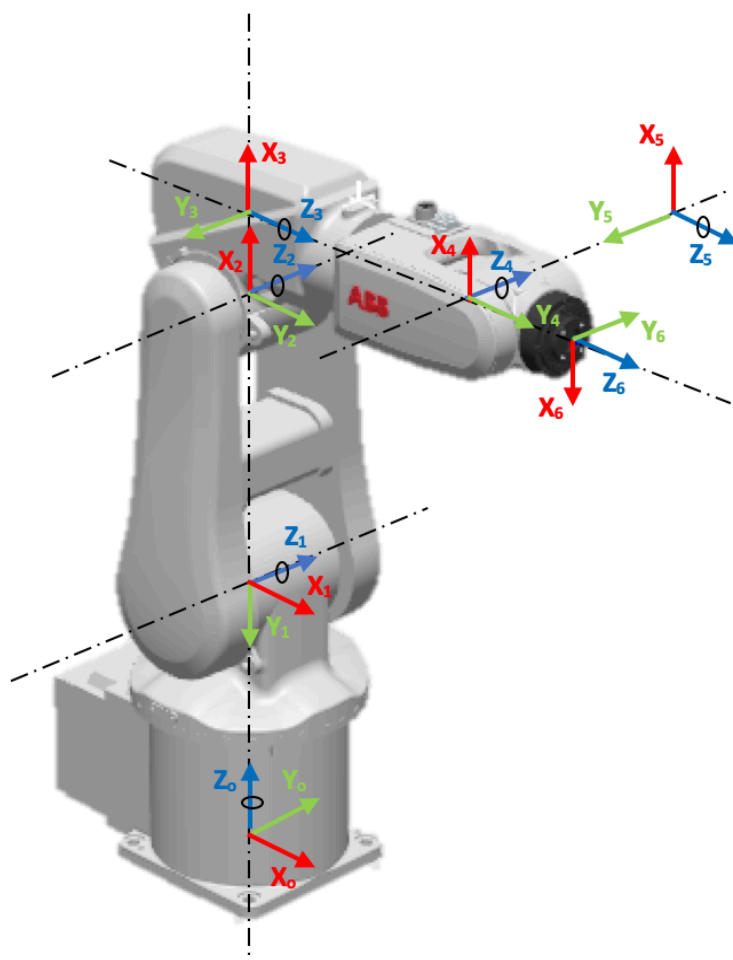


Figura 5.7. Sistemas coordenados del robot IRB 120 según Denavit-Hartenberg



A continuación, utilizando las dimensiones del robot y la ubicación de los sistemas de coordenadas, se determinan los parámetros de Denavit-Hartenberg del robot, recogidos en la siguiente tabla.

Eslabón	$a$ (mm)	$\alpha$ (°)	$d$ (mm)	$\theta$ (°)
1	0	-90	290	$\theta_1$
2	270	0	0	$\theta_2 - 90$
3	70	-90	0	$\theta_3$
4	0	90	302	$\theta_4$
5	0	-90	0	$\theta_5$
6	0	0	72	$\theta_6 - 180$

**Tabla 5.1.** Parámetros Denavit-Hartenberg del robot IRB 120

Una vez calculados los parámetros de cada eslabón se calculan las matrices  ${}^{i-1}\mathbf{T}_i$ , sustituyendo los parámetros D-H por sus correspondientes valores numéricos en la matriz simbólica vista con anterioridad (ver Figura 5.8). Además, para simplificar las matrices de transformación homogéneas intermedias  ${}^1\mathbf{T}_2$  y  ${}^5\mathbf{T}_6$  se han utilizado las siguientes identidades trigonométricas:

$$\text{sen}(\theta - 90) = -\cos(\theta)$$

$$\cos(\theta - 90) = \text{sen}(\theta)$$

$$\text{sen}(\theta - 180) = -\text{sen}(\theta)$$

$$\cos(\theta - 180) = -\cos(\theta)$$

$$\begin{array}{l}
 {}^0\mathbf{T}_1 = \begin{bmatrix} \cos\theta_1 & 0 & -\text{sen}\theta_1 & 0 \\ \text{sen}\theta_1 & 0 & \cos\theta_1 & 0 \\ 0 & -1 & 0 & 290 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^1\mathbf{T}_2 = \begin{bmatrix} \text{sen}\theta_2 & \cos\theta_2 & 0 & 270 \cdot \text{sen}\theta_2 \\ -\cos\theta_2 & \text{sen}\theta_2 & 0 & -270 \cdot \cos\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 {}^2\mathbf{T}_3 = \begin{bmatrix} \cos\theta_3 & 0 & -\text{sen}\theta_3 & 70 \cdot \cos\theta_3 \\ \text{sen}\theta_3 & 0 & \cos\theta_3 & 70 \cdot \text{sen}\theta_3 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^3\mathbf{T}_4 = \begin{bmatrix} \cos\theta_4 & 0 & \text{sen}\theta_4 & 0 \\ \text{sen}\theta_4 & 0 & -\cos\theta_4 & 0 \\ 0 & 0 & 1 & 302 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 {}^4\mathbf{T}_5 = \begin{bmatrix} \cos\theta_5 & 0 & -\text{sen}\theta_5 & 0 \\ \text{sen}\theta_5 & 0 & \cos\theta_5 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^5\mathbf{T}_6 = \begin{bmatrix} -\cos\theta_6 & \text{sen}\theta_6 & 0 & 0 \\ -\text{sen}\theta_6 & -\cos\theta_6 & 0 & 0 \\ 0 & 0 & 1 & 72 \\ 0 & 0 & 0 & 1 \end{bmatrix}
 \end{array}$$

**Figura 5.8.** Matrices de transformación homogénea entre sistemas de coordenadas

Finalmente se obtiene la matriz de transformación  $\mathbf{T}$  que indica la localización del sistema final con respecto al sistema de referencia de la base del robot, como producto del conjunto de matrices:

$$\mathbf{T} = {}^0\mathbf{T}_1 \cdot {}^1\mathbf{T}_2 \cdot {}^2\mathbf{T}_3 \cdot {}^3\mathbf{T}_4 \cdot {}^4\mathbf{T}_5 \cdot {}^5\mathbf{T}_6 = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Como se puede observar, esta matriz está dividida en cuatro partes, de las cuales dos de ellas son muy relevantes. La primera submatriz (3x3) expresa el giro del sistema de coordenadas del extremo del robot respecto la base y la segunda submatriz (3x1) indica la posición del sistemas de coordenadas del extremo del robot respecto a la base del mismo. Las otras dos son el vector de perspectiva (0,0,0) y el factor de escala (1), ya comentadas anteriormente.

Para hacer más ameno el desarrollo de la matriz y más fácil su comprensión se utilizarán simplificaciones en la formulación tales que:

$$\text{Cos}(\theta_1) = C\theta_1 = C_1$$

$$\text{Sen}(\theta_1) = S\theta_1 = S_1$$

$$\text{Cos}(\theta_1 + \theta_2) = C(\theta_1 + \theta_2) = C_{12}$$

$$\text{Sen}(\theta_1 + \theta_2) = S(\theta_1 + \theta_2) = S_{12}$$

A continuación se desarrollan los términos de la matriz:

$$n_x = -C_6(C_5(S_1S_4 + C_4C_1S_{23}) + S_5C_1C_{23}) - S_6(C_4S_1 - S_4C_1S_{23})$$

$$n_y = -C_6(C_5(C_1S_4 - C_4S_1S_{23}) - S_5S_1C_{23}) + S_6(C_4C_1 + S_4S_1S_{23})$$

$$n_z = S_6C_{23}S_4 + C_6(S_{23}S_5 - C_{23}C_4C_5)$$

$$o_x = S_6(C_5(S_1S_4 + C_4C_1S_{23}) + S_5C_1C_{23}) - C_6(C_4S_1 - S_4C_1S_{23})$$

$$o_y = -S_6(C_5(C_1S_4 - C_4S_1S_{23}) - S_5S_1C_{23}) + C_6(C_4C_1 + S_4S_1S_{23})$$

$$o_z = C_6C_{23}S_4 - S_6(S_{23}S_5 - C_{23}C_4C_5)$$

$$a_x = -S_5(S_1S_4 + C_4C_1S_{23}) + C_5C_1C_{23}$$

$$a_y = -S_5(C_1S_4 - C_4S_1S_{23}) + C_5S_1C_{23}$$

$$a_z = C_5S_{23} - C_4S_5C_{23}$$

$$p_x = 72 \cdot (C_5C_1C_{23} - S_5(S_1S_4 + C_4C_1S_{23})) + C_1(70 \cdot S_{23} + 302 \cdot C_{23} + 270 \cdot S_2)$$

$$p_y = 72 \cdot (S_5(C_1S_4 - C_4S_1S_{23}) + C_5S_1C_{23}) + S_1(270 \cdot S_2 + 70 \cdot S_{23} + 302 \cdot C_{23})$$

$$p_z = -302 \cdot S_{23} + 70 \cdot C_{23} + 270 \cdot C_2 - 72 \cdot S_{23}C_5 + 290$$

Finalmente, para implementar este modelo en la interfaz gráfica que se va a explicar en el próximo capítulo, se ha creado una función en *MATLAB* denominada **Modelo\_Geométrico\_Directo** que se puede encontrar en el Anexo III. Esta función permite obtener la matriz de transformación homogénea *T* a partir de los valores de las coordenadas articulares del robot.

### 5.3. Cuaternios.

Existen diversas formas de definir la orientación del extremo del robot, y por tanto de la herramienta anclada a ella: los ángulos de Euler, los ángulos de Roll-Pitch-Yaw, el par de rotación y los cuaternios.

En este caso se ha elegido el uso de **cuaternios**, pues son los que el programa *RobotStudio* utiliza para definir la orientación. Un cuaternio es un sistema sobrep parametrizado formado por cuatro componentes, es decir, utiliza 4 elementos para definir 3 grados de libertad. Además, debe estar normalizado, es decir,

$$q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1.$$

Para calcular sus cuatro componentes se ha hecho uso de la matriz de transformación T calculada anteriormente y se han obtenido las siguientes ecuaciones:

$$q_1 = \frac{1}{2} \cdot \sqrt{n_x + o_y + a_z + 1}$$

$$q_2 = \frac{1}{2} \cdot \sqrt{n_x - o_y - a_z + 1} \quad ; \quad \text{signo} q_2 = \text{signo}(o_z - a_y)$$

$$q_3 = \frac{1}{2} \cdot \sqrt{-n_x + o_y - a_z + 1} \quad ; \quad \text{signo} q_3 = \text{signo}(a_x - n_z)$$

$$q_4 = \frac{1}{2} \cdot \sqrt{-n_x - o_y + a_z + 1} \quad ; \quad \text{signo} q_4 = \text{signo}(n_y - o_x)$$

Como ejemplo, se va a estudiar el cambio de orientación que sufre la herramienta del robot encargado de suturar respecto la base del mismo cuando la incisión se ha realizado en la dirección X (ver Figura 5.9).

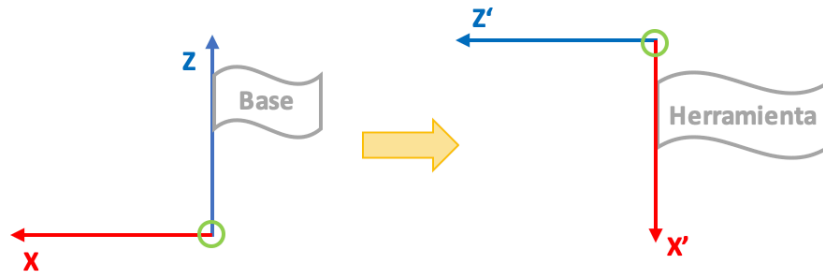


Figura 5.9. Cambio de orientación de la herramienta

$$\text{Base } \mathbf{R}_{\text{Herramienta}} = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

$$q_1 = \frac{\sqrt{2}}{2}$$

$$q_2 = 0$$

$$q_3 = \frac{\sqrt{2}}{2}$$

$$q_4 = 0$$

## 5.4. Matriz de cambio de base.

Debido a la necesidad que se tiene de que los dos robots utilizados en el proyecto alcancen la misma posición para realizar cada uno la acción correspondiente, surge la obligación de calcular una matriz de cambio de coordenadas entre ambos robots. Esto se debe a que cuando se recibe la posición en la que se encuentra uno de los robots, ésta se recibe respecto del sistema de referencia de ese robot y por lo tanto, si esa misma posición se envía al otro robot, este último se movería hacia otro punto diferente.

El objetivo es obtener el vector de posición del segundo robot (respecto de su sistema de coordenadas) por medio del producto de la matriz de cambio de coordenadas del primer robot al segundo y el vector de posición del primer robot respecto de su sistema de coordenadas.

Para calcular dicha matriz hay que tener en cuenta la distribución de la estación de trabajo mostrada en la Figura 5.10, es decir, la distancia a la que se encuentran ambos robots y la orientación de sus sistemas de referencia.

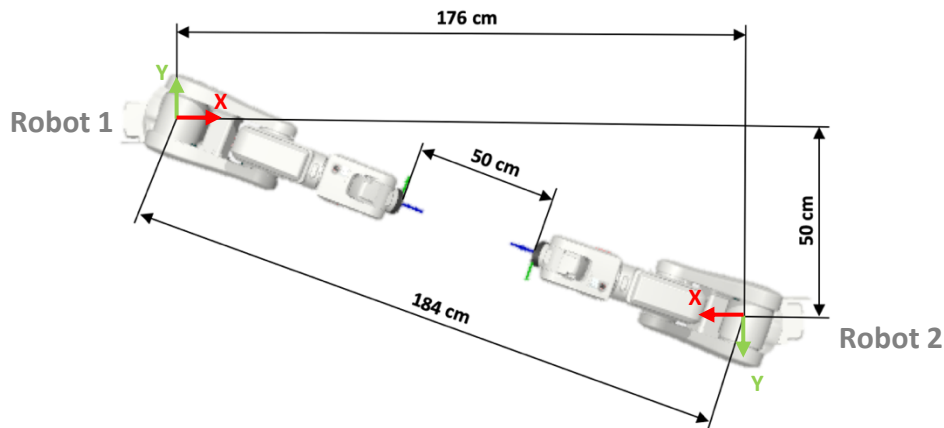


Figura 5.10. Dimensiones estación de trabajo y sistemas de referencia

Entonces, la **matriz de cambio de base del primer robot al segundo** es la siguiente:

$$M_{1-2} = \begin{bmatrix} -1 & 0 & 0 & 1760 \\ 0 & -1 & 0 & -500 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

donde las unidades de la submatriz (3x1) correspondiente a la posición del sistema de coordenadas del segundo robot respecto del primero son milímetros.

Finalmente, para conocer cual tiene que ser el vector posición  $[X_2, Y_2, Z_2]$  del segundo robot para que alcance la posición del primer robot  $[X_1, Y_1, Z_1]$ , se debe proceder de la siguiente forma:

$$\begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 1760 \\ 0 & -1 & 0 & -500 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \end{bmatrix} \Rightarrow \begin{aligned} X_2 &= 1760 - X_1 \\ Y_2 &= -500 - Y_1 \\ Z_2 &= Z_1 \end{aligned}$$

# CAPÍTULO 6. DESARROLLO DE UNA INTERFAZ GRÁFICA DE CONTROL EN MATLAB. GUI.

Con el objetivo de controlar ambos robots desde el cliente, se han diseñado y programado varias interfaces gráficas de usuario GUI (Graphical User Interface) en función de la aplicación quirúrgica a estudiar. Se tratan de unas interfaces sencillas e intuitivas en la se encuentran incorporados todos los movimientos y operaciones de los robots a través de cuadros de texto, menús desplegables y diferentes tipos de botones, todos ellos programados.

## 6.1. Conceptos generales.

Para crear la interfaz gráfica se utiliza el comando **guide** y se selecciona la opción que viene por defecto, *Blank GUI*. Esta selección crea una pantalla vacía en la que posteriormente se implementan los diferentes cuadros de texto y botones que dan forma a la interfaz gráfica. Además, crea directamente una serie de funciones estandarizadas por defecto, una de las cuales (**varargout**) inicializa dicha interfaz (ver Figura 6.1).

```
function varargout = Interfaz_Incision(varargin)
% INTERFAZ_INCISION MATLAB code for Interfaz_Incision.fig
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',   gui_Singleton, ...
                  'gui_OpeningFcn', @Interfaz_Incision_OpeningFcn, ...
                  'gui_OutputFcn',  @Interfaz_Incision_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
```

Figura 6.1. Función de inicialización de la GUI

Tras esta primera función de inicialización, se crean dos más: la función de apertura (**OpeningFcn**) y la función de salida (**OutputFcn**), las cuales se muestran en la Figura 6.2.

En ambas aparecen dos estructuras típicas en las interfaces gráficas, la estructura de datos *hObject* y la estructura de datos *handles*.

- **hObject** hace referencia al identificador del objeto que está actualmente en ejecución, es decir, al valor de la función que se está ejecutando. Cuando la función termine de ejecutarse, *hObject* pasará a tener el valor de la nueva función que se ejecute y, por lo tanto, este valor se actualizará. Es decir, esta estructura de datos no permite almacenar infinitamente el valor de una de las funciones de la interfaz.

- **handles** es la estructura que se utiliza como argumento de entrada en todas las funciones que integran la GUI; por ello, si se necesita pasar una variable del espacio de trabajo de una función a otra dentro de la misma interfaz, se debe usar la siguiente sintaxis para crear una nueva variable con estructura *handles*:

*handles.NombreVariable*

Como la estructura *hObject* es una estructura común para todas las funciones de la interfaz, en caso de querer almacenar el valor del *hObject* de la función que se ha ejecutado, se debe crear una variable *handles* en dicha función y asignarle el valor del *hObject*. De este modo, al ejecutar otra función de la interfaz, el *hObject* cambiará, pero el valor anterior quedará almacenado en la variable *handles* creada.

Para almacenar y actualizar las estructuras *handles*, de forma que posteriormente puedan ser usadas dentro de la interfaz, se utiliza la instrucción **guidata**.

```
% --- Executes just before Interfaz_Incision is made visible.
function Interfaz_Incision_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
% varargin    command line arguments to Interfaz_Incision (see VARARGIN)

% Choose default command line output for Interfaz_Incision
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Interfaz_Incision wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = Interfaz_Incision_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

*Figura 6.2. Función de apertura y salida de la GUI*

A partir de este momento, se empiezan a insertar los distintos botones, cuadros y menús que compondrán la interfaz. Es importante asignar a cada componente **etiquetas** o **tags**, que son propiedades que proporcionan un identificador único. De este modo, va a ser más sencillo identificar cada componente dentro del código de programación, evitando así confusiones innecesarias durante la programación.

Además, cada componente lleva asociadas diferentes funciones dependiendo de su utilidad. Las más recurrentes son las siguientes:

- **Callback:** es una función que se ejecuta cuando el usuario realiza alguna acción sobre el componente (como hacer click sobre un botón o seleccionar una opción de un menú desplegable). Es decir, el código que se incluya en esta función compondrá la tarea principal del componente en cuestión.

- **CreateFcn:** es una función que define las propiedades y características del componente al que está asociada.

En la Figura 6.3 se muestra un ejemplo para entender el funcionamiento de los Callbacks, de los tags y de la estructura de datos *handles*. En dicho ejemplo se ha creado un cuadro de texto y un botón, cuyos tags son: `cuadro_texto` y `presionar_botón` respectivamente. El código procede de la siguiente forma: al presionar el botón aparecerá en el cuadro de texto, inicialmente en blanco, un texto.

```
%Cuando se presiona el botón, cuyo tag es presionar_botón, se ejecuta el código de la  
función presionar_botón_Callback  
function presionar_botón_Callback(hObject, eventdata, handles)  
texto='Hola, esto es un ejemplo';  
set(handles.cuadro_texto,'string',texto); %Escribe el texto en el cuadro de texto cuyo  
tag es cuadro_texto
```

*Figura 6.3. Ejemplo uso tag estructura handles y función Callback*

A continuación, se van a explicar los diferentes escenarios e interfaces creados para cada aplicación quirúrgica de estudio.

## 6.2. Casos de estudio.

- **Escenario I: Interfaz gráfica de control automático de operación incisión-sutura.**

Este primer escenario se basa en la realización de una incisión y su posterior sutura. Posee la interfaz gráfica más sencilla, pero a la vez más rígida de los casos estudiados, debido a que es todo prácticamente automático. El único control manual que existe es el de mover inicialmente el robot que realizará incisión a la posición de inicio de la misma. A partir de ese momento, simplemente basta con introducir la longitud, profundidad y dirección de la incisión y el número de grapas que se desean colocar para que el robot se mueva automáticamente realizando las trayectorias oportunas para completar la tarea.

- **Escenario II: Interfaz gráfica de control manual de aplicación incisión-sutura.**

Este escenario surge a fin de dotar al escenario anterior de una mayor flexibilidad, de forma que se puedan realizar incisiones en direcciones y/o formas diferentes y más complejas a las mostradas en el Escenario I. Así, se ha creado un vector que almacene las posiciones que se desean con el fin de que después el robot reproduzca la trayectoria que componen todos los puntos almacenados. Además, se ha mejorado el control manual de los robots, pudiendo seleccionar la velocidad, precisión y tipo de movimiento deseado en cada caso.

- **Escenario III: Interfaz gráfica de control simultáneo de aplicación drenaje-sutura.**

Este escenario surge con el fin de permitir que los robots realicen tareas de forma simultánea y no de forma secuencial como se daba en los Escenarios I y II. En este caso, la aplicación de estudio ya no es incisión-sutura pues no tendría sentido realizar una incisión y momentáneamente realizar la sutura ya que en dicho caso la intervención quirúrgica en sí sería inexistente. Sin embargo, cuando se realiza una incisión, el cuerpo humano sangra y, para realizar una sutura de la manera más limpia y precisa posible, se

debe drenar el exceso de sangre. En base a esto, se tomó la decisión de realizar esta aplicación.

▪ **Escenario IV: Manipulación de material e instrumental quirúrgico.**

Este escenario surge de la necesidad de utilizar diferentes materiales y herramientas a lo largo de la intervención quirúrgica. Para ello, se necesita que un robot le entregue al otro la herramienta que necesite en cada momento mediante movimientos coordinados. Debido a que en una intervención quirúrgica real estos intercambios de material e instrumental se dan alejados del cuerpo humano para evitar poner en riesgo la salud del paciente, se ha decidido aislar esa parte de la estación de trabajo global para enfocarse solamente en la aplicación a estudiar, pudiendo, posteriormente, incluirla en cualquiera de los otros escenarios.



# CAPÍTULO 7. CONTROL AUTOMÁTICO DE OPERACIÓN INCISIÓN-SUTURA.

Esta interfaz se divide en dos: una relacionada con la operación de incisión y otra relacionada con la operación de sutura. Como se ha mencionado anteriormente, es una interfaz intuitiva y sencilla, pero bastante rígida, en la que con la introducción de unos datos muy concretos se consigue que los robots realicen su cometido de manera automática.

## 7.1. Operación de incisión.

Esta interfaz ha sido desarrollada para llevar a cabo todas las funciones implicadas en la realización de la incisión con el robot correspondiente. En ella, cada función se ha dividido en diferentes paneles que contienen las herramientas necesarias para cumplimentarla (ver Figura 7.1).

Figura 7.1. Escenario I: Interfaz gráfica de la operación de incisión

### 7.1.1. Conexión con el robot.

En este primer panel se encuentra la función de establecer la conexión con el robot correspondiente. Para ello, se ha hecho uso de tres cuadros de texto estático, dos cuadros de texto editable y un botón (ver Figura 7.2).

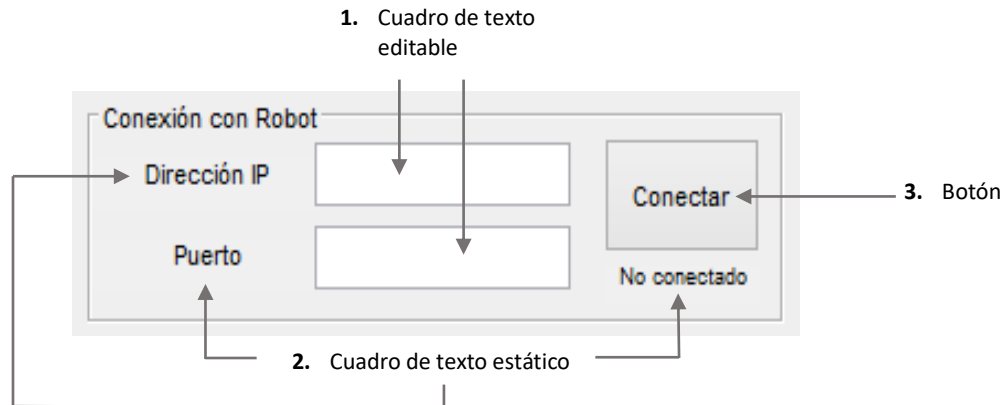


Figura 7.2. Panel: Conexión con Robot

Para poder establecer dicha conexión, es necesario introducir la dirección IP y el puerto del servidor al que se quiere conectar. Estas dos variables serán introducidas en cada uno de los dos cuadros de textos editables y serán almacenadas en el programa gracias a la función **get** y a la estructura de datos *handles*. En la Figura 7.3 se muestra el código ejecutado para procesar la dirección IP del cuadro de texto editable.

```
function direccionIP_1_Callback(hObject, eventdata, handles)
direccionIP_1 = get(handles.direccionIP_1,'string'); %Leer lo que hay en la casilla de
direccion IP y guardarlo en un string
handles.direccionIP_1 = direccionIP_1; %Guardar en estructura handles
guidata(hObject, handles); %Actualizar guidata

% --- Executes during object creation, after setting all properties.
function direccionIP_1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Figura 7.3. Funciones para obtener la dirección IP

La estructura para obtener el número de puerto es similar a la anterior, ya que ambos son cuadros de texto editables. En este caso, se desea obtener el valor numérico de esta variable en vez de la cadena de caracteres (string) y para ello, se utiliza la función **str2double**.

Ya almacenadas estas dos variables, el último paso es configurar el botón de forma que al presionarlo se establezca la conexión. Para ello, se han utilizado los comandos explicados en el apartado 4.2. Además, para confirmar que la conexión se ha recibido de forma satisfactoria, se ha programado un cuadro de texto fijo (tag: estado\_robot1) debajo del botón (ver Figura 7.4). Así, cuando se establezca la conexión aparecerá el mensaje 'Conectado' en vez de 'No conectado'.

```
% --- Executes on button press in conectar_1.
function conectar_1_Callback(hObject, eventdata, handles)
tc_1=tcipip(handles.direccionIP_1,handles.puerto_1); %Datos conexión
handles.tc_1 = tc_1; %Guardar variable en estructura handles
fopen(handles.tc_1); % Habilitar la comunicación
set(handles.estado_robot1,'string','Conectado');
guidata(hObject, handles);
```

Figura 7.4. Función botón Conectar

### 7.1.2. Mover el robot a la posición para empezar la incisión.

La función de este panel es mover el robot a la posición inicial para empezar la incisión, es decir, a una zona muy próxima al cuerpo humano. Debido a la poca flexibilidad que posee la interfaz, se han clasificado los movimientos del robot en dos tipos en función de la velocidad de movimiento. El primero de ellos incluye los movimientos que permiten al robot alcanzar una posición cercana al cuerpo humano, que se realizan a una velocidad de 50 mm/s, mientras que el segundo tipo engloba los diferentes posibles movimientos involucrados de la trayectoria de la incisión que, debido a la necesidad de una precisión muy buena, se realizan a una velocidad de 10 mm/s.

Para el control del movimiento del robot se ha decidido utilizar coordenadas articulares en vez de coordenadas cartesianas debido a que es un tipo de movimiento más rápido y visual.

En este panel (ver Figura 7.5) se ha optado por utilizar botones + y - para cada una de las coordenadas articulares, un botón para iniciar el movimiento y otro para finalizarlo y un grupo de botones de opción para seleccionar los grados que se desean girar en cada movimiento.

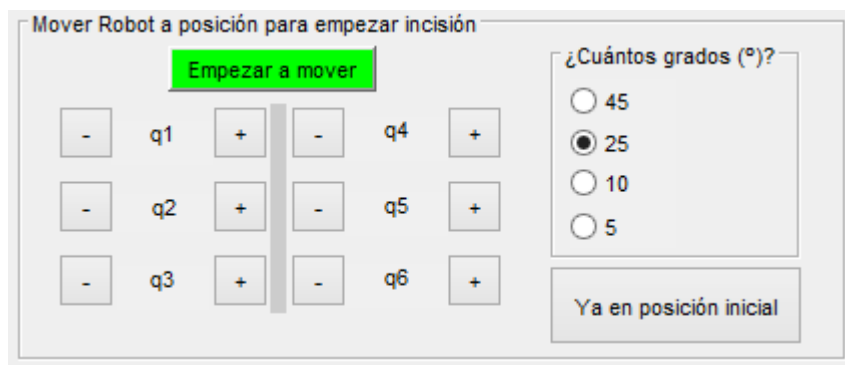


Figura 7.5. Panel: Mover robot a posición para empezar incisión

Para hacer un uso correctamente de este panel, primero se debe presionar el botón 'Empezar a mover' con el que se envía un mensaje al servidor al que esté conectado con la palabra 'Mover' para que dicho servidor interprete la función que debe a realizar. Además, se designan las seis variables articulares de la posición inicial del robot.

En el texto que aparece en este botón pueden aparecer 3 frases diferentes:

- Empezar a mover: cuando aún no se ha presionado el botón y por lo tanto no se ha ejecutado la función.

- Moviendo... : cuando se ha presionado el botón y está permitido el movimiento de las coordenadas articulares mediante sus respectivos botones + y -.
- Fin movimiento: cuando el robot está en la posición para empezar la incisión y se ha presionado el botón 'Ya en posición inicial'.

Después, se debe seleccionar cuántos grados se desea mover. Para ello, se ha programado el panel de botón de grupo que permite que solo se pueda seleccionar uno de ellos. En este caso, la función utilizada no es Callback sino **SelectionChangedFcn**, que permite detectar el valor del botón seleccionado cuando se produce un cambio de selección en ellos. Para saber cual de los cuatro botones se ha seleccionado, se creará una variable *handles* en la que se almacene el valor seleccionado, es decir, el *hObject* de la función.

A continuación, se presionan los botones + y - de las coordenadas articulares que se deseen mover. La programación de estos botones consiste en actualizar las variables articulares en función del valor de grados seleccionado y, una vez estén actualizadas, ejecutar la función creada aparte de la GUI denominada *Modelo\_Geométrico\_Directo* que permite obtener el **modelo geométrico directo** del robot (explicado en apartado 5.2).

Por último, una vez esté el robot en la posición en la que se desea empezar la incisión se debe presionar el botón 'Ya en posición inicial', que enviará un mensaje al servidor para que entienda que la función completa de este panel ha terminado de ejecutarse. Además, como se ha comentado anteriormente, el texto del primer botón cambiará a 'Fin movimiento'.

En la Figura 7.6 se muestra un fragmento del código de este panel. Cabe destacar que solo se muestra el código de la función Callback del botón positivo (+) de la coordenada articular  $q_1$ , pues el código de los botones restantes puede ser extrapolado simplemente cambiando el signo y el nombre de la coordenada articular según corresponda.

```
% --- Executes on button press in empezar_mover.
function empezar_mover_Callback(hObject, eventdata, handles)
fwrite(handles.tc_1,'Mover');
set(handles.empezar_mover,'string','Moviendo...');
handles.q1=-105;    handles.q2=-90;    handles.q3=0;    handles.q4=0;    handles.q5=30;
handles.q6=-180;
guidata(hObject,handles);

% --- Executes when selected object is changed in elegir_grados.
function elegir_grados_SelectionChangedFcn(hObject, eventdata, handles)
handles.valor_mover=hObject; %Almacena el valor que está seleccionado en la variable
handles.valor_mover
guidata(hObject,handles);

function q1_positivo_Callback(hObject, eventdata, handles)
if handles.valor_mover==handles.mover_45 %Si está elegido la opción 45°
    handles.q1=handles.q1+45;
elseif handles.valor_mover==handles.mover_25 %Si está elegido la opción 25°
    handles.q1=handles.q1+20;
elseif handles.valor_mover==handles.mover_10 %Si está elegido la opción 10°
    handles.q1=handles.q1+10;
else handles.q1=handles.q1+5; %Si está elegido la opción 5°
end
```

```

handles.robot=1; %Indica que se quiere mover el robot 1 (de incisión) y no el robot 2
(de sutura)
guidata(hObject,handles);
Modelo_Geometrico_Directo(hObject);
guidata(hObject,handles);

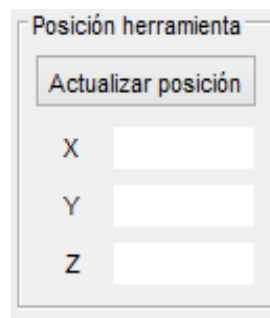
function terminar_mover_Callback(hObject, eventdata, handles)
fwrite(handles.tc_1,'Posicion inicial');
set(handles.empezar_mover,'string','Fin movimiento');
guidata(hObject,handles);

```

**Figura 7.6.** Fragmento de la función panel Mover robot a posición para empezar incisión

### 7.1.3. Actualizar posición de la herramienta.

Este panel tiene la función de mostrar cuál es la posición actual de la herramienta. Es muy simple y consta de un botón y cuadros de texto estático en los que aparecerá el valor de cada una de las coordenadas cuando el cliente reciba la posición del servidor (ver Figura 7.7).



**Figura 7.7.** Panel: Actualizar posición de la herramienta

Para utilizar este panel solo hay que presionar el botón 'Actualizar posición', que enviará un mensaje al servidor para que empiece a recibir los mensajes así como las longitudes de los mismos. En este caso, los mensajes recibidos son los valores de cada coordenada en la que está situada la herramienta. Una vez recibida toda la información, escribirá en cada cuadro de texto el valor de la coordenada que corresponda (ver Figura 7.8).

```

% --- Executes on button press in actualizar_posicion.
function actualizar_posicion_Callback(hObject, eventdata, handles)
global coordenadaX coordenadaY coordenadaZ; %Para poder utilizarlas en otra interfaz
fwrite(handles.tc_1,'Enviar Posicion Herramienta');
longitudX=str2double(char(fread(handles.tc_1,1)));
longitudY=str2double(char(fread(handles.tc_1,1)));
longitudZ=str2double(char(fread(handles.tc_1,1)));
handles.coordenadaX=char(fread(handles.tc_1,[1,longitudX]));
handles.coordenadaY=char(fread(handles.tc_1,[1,longitudY]));
handles.coordenadaZ=char(fread(handles.tc_1,[1,longitudZ]));
coordenadaX=str2num(handles.coordenadaX);
coordenadaY=str2num(handles.coordenadaY);
coordenadaZ=str2num(handles.coordenadaZ);
set(handles.coordenada_X,'string',handles.coordenadaX);
set(handles.coordenada_Y,'string',handles.coordenadaY);
set(handles.coordenada_Z,'string',handles.coordenadaZ);
guidata(hObject,handles);

```

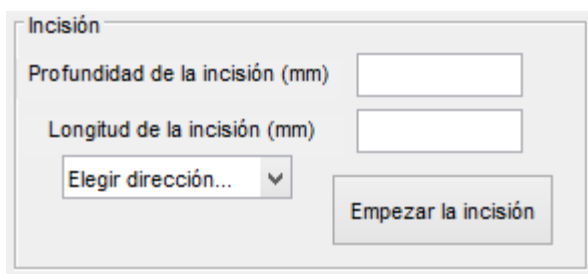
**Figura 7.8.** Función panel Actualizar posición de la herramienta

Como se puede observar, se han utilizado **variables globales** para cada una de las coordenadas. Esto se debe a que las variables globales permiten ser usadas en cualquiera de las interfaces existentes y no solo en la que se han creado. Entonces, como es necesario utilizar los valores de las coordenadas más adelante en otra interfaz (de sutura), se han creado variables globales para estas coordenadas.

#### 7.1.4. Incisión.

Este panel es el último de esta interfaz y se corresponde con la realización de la incisión (ver Figura 7.9). Como la incisión se desea efectuar de la manera más precisa posible, la velocidad de los movimientos del robot se ha reducido todavía más con respecto a la velocidad utilizada para mover al robot a la posición para empezar a suturar (apartado 6.2.2) y es de 10 mm/s.

En este caso, hay que introducir la profundidad deseada de la incisión y la longitud de la misma en dos cuadros de texto editable. Además, hay un menú desplegable en el que aparecen todas las direcciones posibles según las que efectuar la incisión, que son: el eje X, el eje Y, y los ejes del plano bisector a XY. Así, dado que en cualquier caso la incisión se podrá dar en ambos sentidos, el **número total de direcciones es 8**. Finalmente, al presionar el botón '*Empezar la incisión*', el robot recogerá toda esta información y realizará la incisión según esas condiciones.



*Figura 7.9. Panel: Incisión*

Las funciones Callback de estos componentes son similares a las explicadas anteriormente, por lo que, para evitar repeticiones innecesarias, se vuelven a comentar.

En la Figura 7.10 se muestra una sub-función dentro de la función Callback del botón '*Empezar la incisión*', denominada ***Incisión*** y creada aparte de la GUI, que tiene el código para mover al robot a lo largo de la trayectoria de la incisión. Esta trayectoria consta de **cuatro posiciones** diferentes que se explican a continuación:

- El primer punto se encuentra a 20 mm por encima de la posición inicial (respecto del eje Z).
- Después, la herramienta baja hasta la posición inicial y continúa bajando tantos mm como se hayan introducido en el cuadro de texto de la profundidad de la incisión.
- A continuación, el robot se mueve en la dirección seleccionada en el panel de incisión tantos mm como se haya introducido en el cuadro de texto de la longitud de la incisión.
- Finalmente, el robot sube 80 mm con respecto a la última posición para alejar la herramienta del cuerpo humano.

Cabe pensar que el primer punto no es necesario ya que lo único que hace es subir la herramienta para después bajarla, pero *RobotStudio* no permite realizar una trayectoria entre dos puntos que están a una distancia tan ínfima como es la profundidad típica de una incisión.

```
function [handles] = Incision(hObject)
handles = guidata(hObject);

handles.coordenadaX = str2num(handles.coordenadaX);
handles.coordenadaY = str2num(handles.coordenadaY);
handles.coordenadaZ = str2num(handles.coordenadaZ);
X = char(num2str(handles.coordenadaX));
Y = char(num2str(handles.coordenadaY));
Z = char(num2str(handles.coordenadaZ));
Zinicial = char(num2str(handles.coordenadaZ+20));
Zmedio = char(num2str(handles.coordenadaZ-handles.profundidad));
Zfinal = char(num2str(handles.coordenadaZ+80));

if handles.direccion == 2 %Si la dirección es +X

    Xfinal = char(num2str(handles.coordenadaX+handles.longitud));

    posicion_inicial=[X,',',Y,',',Zinicial];
    fwrite(handles.tc_1, posicion_inicial);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_1= [X, ',', Y, ',', Zmedio];
    fwrite(handles.tc_1, posicion_1);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_2 = [Xfinal, ',', Y, ',', Zmedio];
    fwrite(handles.tc_1, posicion_2);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_final = [Xfinal, ',', Y, ',', Zfinal];
    fwrite(handles.tc_1, posicion_final);

elseif handles.direccion == 3 %Si la dirección es +45° +X

    Xfinal = char(num2str(handles.coordenadaX+(handles.longitud*cosd(45))));
    Yfinal = char(num2str(handles.coordenadaY+(handles.longitud*sind(45))));

    posicion_inicial=[X,',',Y,',',Zinicial];
    fwrite(handles.tc_1, posicion_inicial);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_1= [X, ',', Y, ',', Zmedio];
    fwrite(handles.tc_1, posicion_1);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_2 = [Xfinal, ',', Yfinal, ',', Zmedio];
    fwrite(handles.tc_1, posicion_2);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_final = [Xfinal, ',', Yfinal, ',', Zfinal];
    fwrite(handles.tc_1, posicion_final);

. . .
```

**Figura 7.10.** Fragmento de la función *Incisión*

Como se puede observar, la función **Incisión** de la Figura 7.10 no está completa. Esto se debe a que es una función muy extensa, habiéndose expuesto en esta figura lo fundamental para comprender como están programados los cuatro posiciones de trayectoria en función de la dirección deseada. No obstante, la función completa, se puede encontrar en el Anexo III.

#### 7.1.5. Funciones independientes.

Finalmente, para terminar de crear esta primera interfaz, se han añadido tres botones más. El primero que sirve para mover al robot a la posición de reposo, es decir, la posición en la que estaba antes de empezar la intervención; el segundo es un botón de 'Stop' para que, en caso de emergencia, el programa se pare de inmediato. Finalmente, el tercero es un botón que se hará visible solamente cuando la incisión haya finalizado, cerrando la interfaz de incisión y abriendo la interfaz de sutura.

En la Figura 7.11 se muestran las funciones Callback de los botones 'Stop' e 'Ir a la interfaz de suturar'.

```
% --- Executes on button press in stop.
function stop_Callback(hObject, eventdata, handles)
    fwrite(handles.tc_1, 'Stop');
    fclose(handles.tc_1);
    close(Interfaz_Incision);
    guidata(hObject, handles);

% --- Executes on button press in ir_suturar.
function ir_suturar_Callback(hObject, eventdata, handles)
    fwrite(handles.tc_1, 'Stop');
    fclose(handles.tc_1);
    close(Interfaz_Incision);
    Interfaz_Sutura;
```

**Figura 7.11.** Función Callback botones 'Stop' e 'Ir a interfaz de suturar'



## 7.2. Operación de sutura.

Esta interfaz recoge todas las funciones que suponen la realización de la sutura con el robot correspondiente (ver Figura 7.12). Al igual que en el caso de la interfaz gráfica de incisión, cada función se ha dividido en diferentes paneles.

Figura 7.12. Escenario I: Interfaz gráfica de la operación de sutura

### 7.2.1. Conexión con el robot.

Este panel es idéntico al explicado en la interfaz gráfica de incisión (apartado 6.2.1) puesto que el objetivo es únicamente establecer la conexión entre la interfaz y el robot. Por lo tanto, se deberán introducir los datos de la dirección IP y del puerto correspondientes al robot que se desee utilizar para esta conexión. En este caso, el robot con el que hay que establecer la conexión es diferente al del apartado 6.2.1 y, por lo tanto, los datos de la dirección IP y del puerto cambiarán. Además, el formato del panel se puede observar en la Figura 7.2.

### 7.2.2. Mover robot a posición cercana al cuerpo humano.

Como se puede observar en la Figura 7.13, este panel consta únicamente de un botón, que, en caso de ser presionado, hace que la herramienta del robot se mueva hasta una posición cercana al cuerpo humano para, a partir de ahí, empezar a suturar. Este movimiento se realizará gracias a la función **Modelo\_Geometrico\_Directo** mencionada con anterioridad.

Figura 7.13. Panel: Mover robot a posición cercana al cuerpo humano

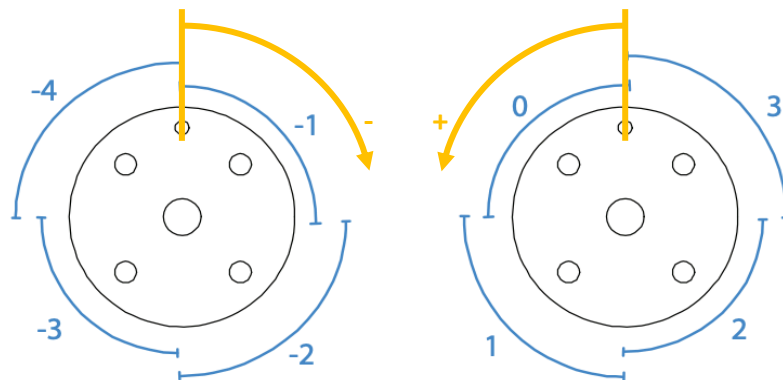
En ocasiones, para alcanzar alguna de las posiciones de la trayectoria de sutura, el robot debía hacer giros de la herramienta y movimientos poco naturales, con la consecuente falta de seguridad que eso conlleva para el ser humano que está siendo sometido a la intervención. Entonces, se decidió cambiar configuración inicial del robot para que pudiese alcanzar las posiciones de manera natural independientemente de la dirección en la que se debía sutura.

Un robot puede alcanzar la misma posición, situando la herramienta en la misma orientación y posición, a partir de distintas posiciones o configuraciones de los ejes del robot. *RobotStudio* permite conocer el número de configuraciones posibles para una posición.

Realizando una serie de pruebas, se llegó a la conclusión de que la **configuración idónea** para toda la trayectoria del robot de incisión era la siguiente: **[-1, 1, -2, 0]**. Los tres primeros componentes de la configuración reflejan el intervalo en el que los ejes 1, 4 y 6 respectivamente están girados respecto a su posición neutral.

- El primer componente de la configuración elegida (**-1**) indica que el eje 1 del robot está girado unos grados dentro del intervalo  $[0, -90]$  respecto de la posición neutral.
- El segundo componente (**1**) indica que el eje 4 está girado unos grados dentro del intervalo  $[90, 180]$  respecto de la posición neutral.
- Asimismo, el tercer componente (**-2**) indica que el eje 6 está girado unos grados dentro del intervalo  $[-90, -180]$  respecto de la posición neutral.
- Finalmente, el último componente (**0**) depende del tipo de robot.

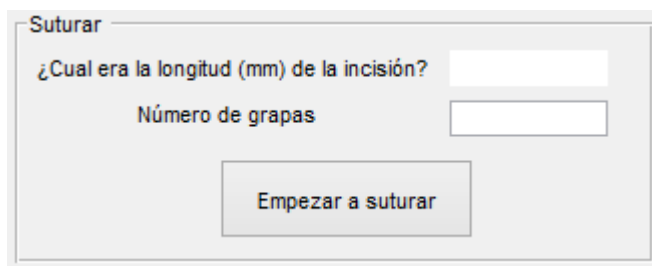
La Figura 7.14 muestra los cuadrantes de configuración para el eje 1, 4 o 6 en un robot de 6 ejes, donde la posición cero es recta.



*Figura 7.14. Cuadrantes de configuración para el eje 1, 4 y 6 de un robot de 6 ejes*

### 7.2.3. Suturar.

Este panel permite recopilar los datos necesarios para llevar a cabo la sutura de la incisión (ver Figura 7.15).



**Suturar**  
 ¿Cual era la longitud (mm) de la incisión?   
 Número de grapas   
 Empezar a suturar

*Figura 7.15. Panel: Suturar*

En una intervención quirúrgica real primero se realiza la incisión, luego tiene lugar la propia intervención como tal y finalmente se sutura; es decir, puede pasar un periodo de tiempo considerablemente largo en el que se pierda constancia de la longitud de la incisión que se ha realizado, y por ende, del número de grapas que se necesitan utilizar. Por eso, en este panel se ha añadido un cuadro de texto fijo en el que cuando se presione el botón ‘Conectar’ del panel *Conexión con robot* (apartado 6.3.1) aparecerá la longitud de la incisión introducida en la interfaz de incisión a modo de recordatorio.

En el cuadro de texto editable se debe introducir el número de grapas que se desean utilizar a lo largo de la sutura y el botón sirve para iniciar la acción de suturar.

Finalmente, al presionar el botón ‘Empezar a suturar’ se ejecutarán una serie de comandos, de entre los cuales destaca una función creada aparte de la GUI, denominada **Sutura**. Esta función contiene las diferentes posiciones de la trayectoria que debe realizar la herramienta para suturar en función de la dirección en que se realice. Esta trayectoria consta de **3·n posiciones**, siendo n el número de grapas, y son las siguientes:

- La primera posición se encuentra elevada 50 mm en el eje Z respecto a la posición en la que se inició la incisión.
- La segunda posición se corresponde con la posición inicial. Como el TCP de la herramienta de sutura no está en el extremo de la pinza, sino en el centro de la misma, esto permitirá que se ejerza una ligera presión sobre la piel para poder agarrar correctamente la piel para suturar. Además, en esta posición la pinza se cerrará durante 2 segundos y se volverá a abrir simulando que se ha colocado una grapa.
- La tercera posición es idéntica a la primera posición.
- El resto de posiciones siguen el mismo patrón en función del número de grapas a lo largo de la incisión y, por tanto, de la separación entre ellas.

A continuación, en la Figura 7.16 se muestra un fragmento de la función **Sutura** para entender lo explicado anteriormente. Al igual que se ha comentado anteriormente en el apartado 6.2.4, se ha mostrado un fragmento para facilitar la comprensión, pudiendo encontrarse la función completa en el Anexo III.

```
function [handles] = Sutura(hObject)
handles = guidata(hObject);

global longitud direccion coordenadaX coordenadaY coordenadaZ;
handles.distancia_grapas = longitud/handles.grapas;

for i=1:handles.grapas

    X = char(num2str(coordenadaX));
    Y = char(num2str(coordenadaY));
    Z = char(num2str(coordenadaZ));
    Zarriba = char(num2str(coordenadaZ+50));

    if direccion == 2 %Si la dirección es +X

        X = char(num2str(coordenadaX+((i-1)*handles.distancia_grapas)));

        posicion_inicial=[X,',',Y,',',Zarriba];
        fwrite(handles.tc_2, posicion_inicial);
        mensaje=fread(handles.tc_2,2);

        waitfor(mensaje,'Ok');
        posicion_medio=[X,',',Y,',',Z];
        fwrite(handles.tc_2, posicion_medio);
        mensaje=fread(handles.tc_2,2);
        waitfor(mensaje,'Ok');
        posicion_final=[X,',',Y,',',Zarriba];
        fwrite(handles.tc_2, posicion_final);
        mensaje=fread(handles.tc_2,2);
        waitfor(mensaje,'Ok');

    elseif direccion == 3 %Si la dirección es 45° +X

        X = char(num2str(coordenadaX+((i-1)*handles.distancia_grapas*cosd(45))));
        Y = char(num2str(coordenadaY+((i-1)*handles.distancia_grapas*sind(45))));

        posicion_inicial=[X,',',Y,',',Zarriba];
        fwrite(handles.tc_2, posicion_inicial);
        mensaje=fread(handles.tc_2,2);
        waitfor(mensaje,'Ok');
        posicion_medio=[X,',',Y,',',Z];
        fwrite(handles.tc_2, posicion_medio);
        mensaje=fread(handles.tc_2,2);
        waitfor(mensaje,'Ok');
        posicion_final=[X,',',Y,',',Zarriba];
        fwrite(handles.tc_2, posicion_final);
        mensaje=fread(handles.tc_2,2);
        waitfor(mensaje,'Ok');

    . . .
end
```

*Figura 7.16. Fragmento de la función Sutura*

Como se puede observar en la función, se hace uso de las **variables globales** comentadas anteriormente en el apartado 7.1.3. Esto se debe a que dichas variables pertenecen a la interfaz gráfica de incisión y, al ser necesario su uso en la interfaz gráfica de sutura, se deben convertir en variables globales.

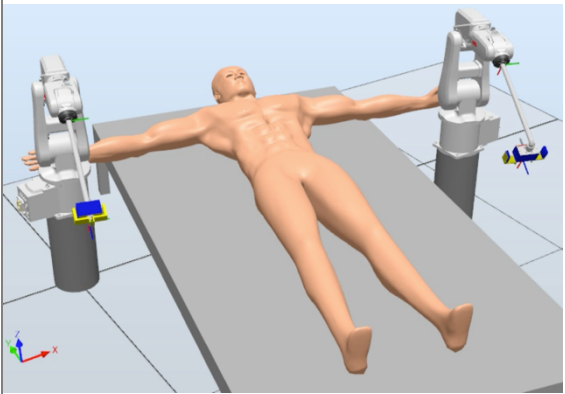
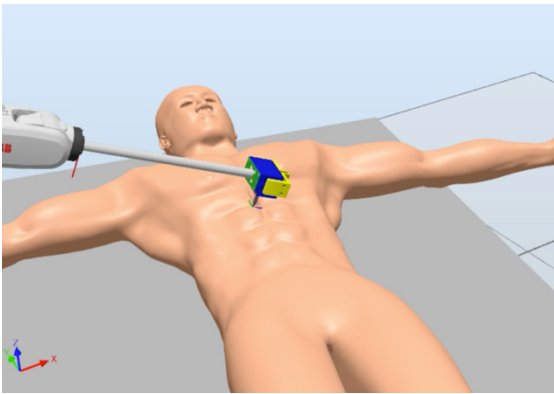
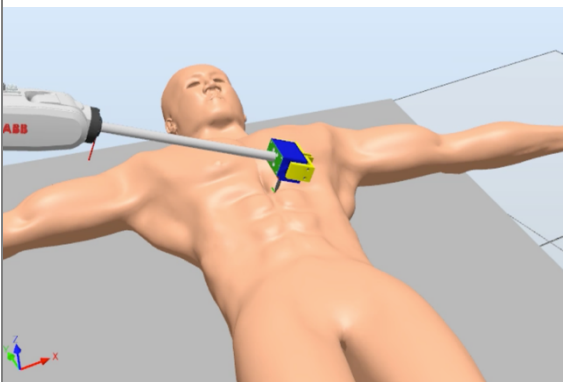
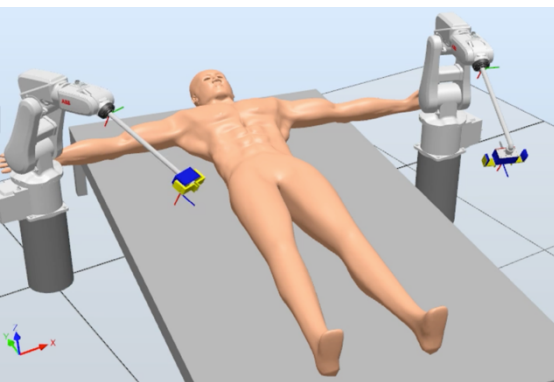
Como se puede observar en el código, no existe ninguna línea asociada a la funcionalidad de la pieza. Esto se debe a que dicha funcionalidad está implementada en la función Sutura correspondiente a *RobotStudio* que, a su vez, recibe las posiciones enviadas por la función expuesta. Esta función de *RobotStudio* se puede consultar en el Anexo IV.

#### 7.2.4. Funciones independientes.

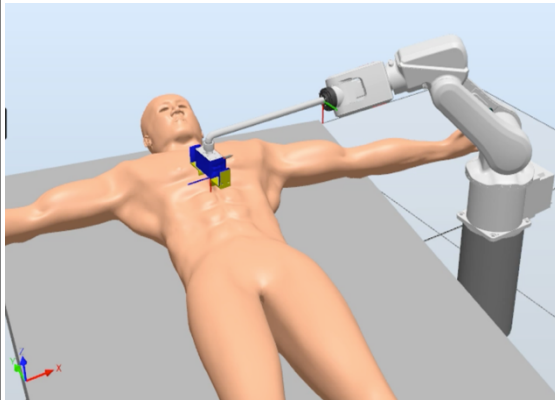
Finalmente, y al igual que en la interfaz de incisión, se han creado tres botones. El primero sirve para volver a la posición de reposo; el segundo es un botón de 'Stop' que detendrá inmediatamente el programa en caso de emergencia. Finalmente, el tercero es un botón para finalizar la intervención, que se hará visible solamente cuando la sutura haya finalizado.

### 7.3. Esquema de las etapas de la simulación.

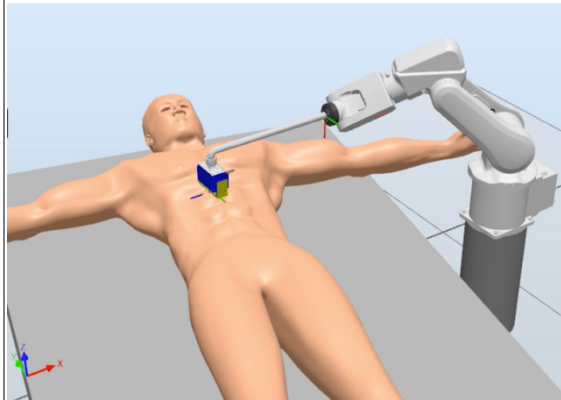
En este apartado se realiza un esquema con las diferentes etapas de la simulación de este escenario con el fin de que se pueda entender con mayor claridad como se ejecutan todas las funciones explicadas anteriormente (ver Tabla 7.1).

<p><b>1. Posición inicial</b></p> 	<p><b>2. Mover manualmente primer robot a la posición de inicio de incisión e introducir longitud y profundidad de incisión</b></p> 
<p><b>3. Realización de incisión</b></p> 	<p><b>4. Primer robot vuelve a posición inicial</b></p> 

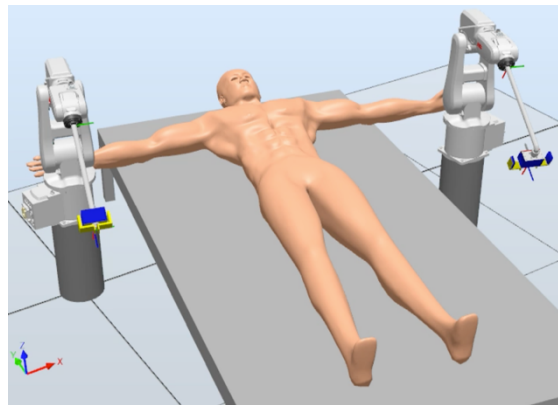
5. Segundo robot se mueve automáticamente a zona de suturar



6. Suturar en los puntos calculados en función de la longitud de la incisión y del número de grapas introducido



7. Volver a la posición inicial



**Tabla 7.1.** Esquema etapas simulación Escenario I

## CAPÍTULO 8. CONTROL MANUAL DE OPERACIÓN INCISIÓN-SUTURA.

La interfaz de este escenario también se ha desarrollado para llevar a cabo el control de la operación de incisión y sutura, pero en este caso de una manera completamente manual. Como ya se ha comentado anteriormente, este escenario surge de la necesidad de una mayor flexibilidad en los movimientos del robot debido a que la alta rigidez del Escenario I. En este caso, la interfaz es un poco más compleja que la anterior debido a que esta permitido la elección de muchas más variables en lo que se refiere a movimiento que en el caso del control automático. Esta interfaz también se divide en dos, una para la operación de incisión y otra para la operación de sutura.

### 8.1. Operación de incisión.

Al igual que en el Escenario I, la interfaz se ha dividido en diferentes paneles según la función que deben realizar (establecer la conexión, mover manualmente el robot, realizar la incisión). En la Figura 8.1 se muestra la interfaz correspondiente a la operación de incisión.

Figura 8.1. Escenario II: Interfaz gráfica de la operación de incisión

Como se puede observar, hay cambios apreciables con respecto a la interfaz del Escenario I (apartado 7.1), los cuales permiten una mayor personalización de la aplicación según las necesidades existentes.

El panel de Conexión con Robot y los botones independientes son similares a los expuestos en el Escenario I y, para evitar repeticiones innecesarias no van a ser comentados en detalle.

### 8.1.1. Configuración del movimiento.

Para dotar a la aplicación de estudio de la flexibilidad comentada anteriormente, se ha creado un panel para configurar todos los parámetros de movimiento del robot antes de que dicho movimiento tenga lugar (ver Figura 8.2). De esta nueva forma se consigue una total libertad para que cada movimiento del robot sea único y personalizado, contrario al caso mostrado en el Escenario I, donde existía una evidente falta de eficiencia al darse la misma velocidad en los movimientos cercanos al cuerpo humano y en los alejados de este.

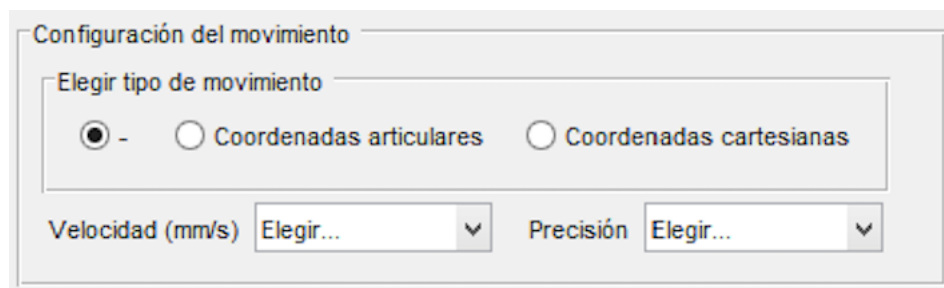


Figura 8.2. Panel: Configuración del movimiento

En primer lugar, se debe elegir el tipo de coordenadas en las que se desea mover al robot (articulares o cartesianas). En este caso no se ha usado la función Callback sino **SelectionChangeFcn**, lo que permite que cuando se seleccione una de ellas, aparezca el panel correspondiente a ese tipo de coordenadas, antes invisible. Además, cuando se quiera cambiar el tipo de coordenadas en las que realizar el movimiento, se hará invisible el panel de coordenadas actual y aparecerá el de las nuevas (ver Figura 8.3). Es decir, en todo momento tendremos únicamente un panel de movimiento del robot visible.

```
% --- Executes when selected object is changed in elegir_movimiento.
function elegir_movimiento_SelectionChangedFcn(hObject, eventdata, handles)
if hObject==handles.articulares
    handles.movimiento='1';
    set(handles.mover_cartesianas,'Visible','Off');
    set(handles.mover_articulares,'Visible','On');
elseif hObject==handles.cartesianas
    handles.movimiento='2';
    set(handles.mover_articulares,'Visible','Off');
    set(handles.mover_cartesianas,'Visible','On');
end
guidata(hObject, handles);
```

Figura 8.3. Función Elegir tipo de movimiento

En segundo lugar, en los menús desplegables se seleccionan la velocidad y la precisión con la que se quiere realizar el movimiento. Estos menús desplegables actúan como un vector, es decir, cada una de las opciones que se pueden seleccionar tienen un valor asociado en función de la posición en la que se encuentren. Si se obtiene ese valor asociado, el programa interpretará a que velocidad y precisión se corresponde (ver Figura 8.4).



```

% --- Executes on selection change in elegir_velocidad.
function elegir_velocidad_Callback(hObject, eventdata, handles)
handles.elegir_velocidad=get(hObject, 'Value');
if handles.elegir_velocidad==2
    handles.velocidad='5';
elseif handles.elegir_velocidad==3
    handles.velocidad='10';
elseif handles.elegir_velocidad==4
    handles.velocidad='20';
elseif handles.elegir_velocidad==5
    handles.velocidad='50';
elseif handles.elegir_velocidad==6
    handles.velocidad='100';
else handles.velocidad='200';
end
guidata(hObject, handles);

```

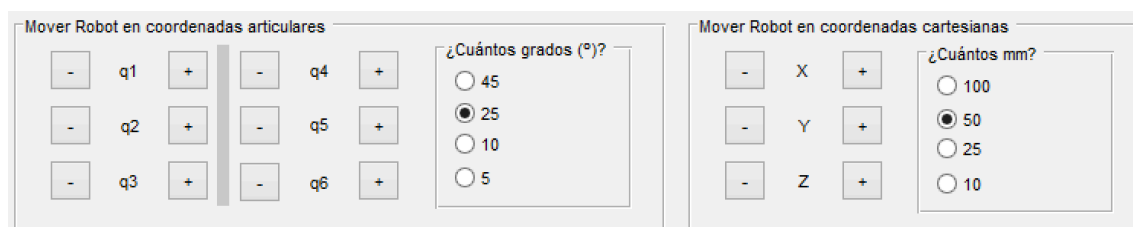
**Figura 8.4.** Función Elegir velocidad

En la Figura 8.4 solo se muestra la función Callback para elegir la velocidad de movimiento. La función correspondiente a elegir la precisión no se ha mostrado, pues el código es muy similar.

### 8.1.2. Mover Robot en coordenadas articulares y cartesianas.

Estos paneles guardan una estrecha relación con el mostrado anteriormente en el apartado 8.1.1. Como se ha comentado, cada uno de estos dos paneles aparecerá según se haya seleccionado el botón de coordenadas articulares o cartesianas.

En ambos paneles se tienen botones + y - para cada una de las coordenadas articulares y cartesianas. Así como un grupo de botón de opciones para seleccionar los grados o milímetros que se desean mover según corresponda (ver Figura 8.5).



**Figura 8.5.** Panel: Mover Robot en coordenadas articulares/cartesianas

Los robots utilizados tienen 6 grados de libertad, por lo que para definirlos completamente se necesitan 6 coordenadas. Como se puede observar, mientras que en el panel de coordenadas articulares aparecen 6 coordenadas articulares, el panel de coordenadas cartesianas solo aparecen las coordenadas X,Y,Z (faltando los tres ángulos que definen la orientación). Esto se debe a la longitud de la herramienta que, con cualquier mínimo giro en coordenadas cartesianas da un error de movimiento no realizable. Además, este cambio de orientación en coordenadas cartesianas sería necesario en caso de tener una herramienta no simétrica, como por ejemplo, un bisturí real, el cual tiene una parte más afilada que otra y, por lo tanto la parte afilada tendría que estar en contacto con la piel. En este caso, al ser la herramienta un punzón, no es imprescindible que incida de una determinada manera y, por lo tanto, se ha decidido no aplicarlo al panel.

La programación de ambos grupos de botón de opciones y la configuración de los botones de las coordenadas articulares es idéntica a la explicada en el Escenario I (apartado 7.1.2).

En la Figura 8.6 se muestra el código de uno de los botones de las coordenadas cartesianas; el resto se pueden extrapolar de él cambiando la coordenada y el signo.

```
function x_positivo_Callback(hObject, eventdata, handles)
global coordenada_x coordenada_y coordenada_z;
Configuracion_Movimiento(hObject);
if handles.mm==handles.mover_100mm
    coordenada_x=coordenada_x+100;
elseif handles.mm==handles.mover_50mm
    coordenada_x=coordenada_x+50;
elseif handles.mm==handles.mover_25mm
    coordenada_x=coordenada_x+25;
else coordenada_x=coordenada_x+10;
end
guidata(hObject,handles);
X=char(num2str(coordenada_x));
Y=char(num2str(coordenada_y));
Z=char(num2str(coordenada_z));
posicion = [X, ',', Y, ',', Z];
fwrite(handles.tc_1, posicion);
Obtener_Coordenadas_Articulares(hObject);
guidata(hObject,handles);
```

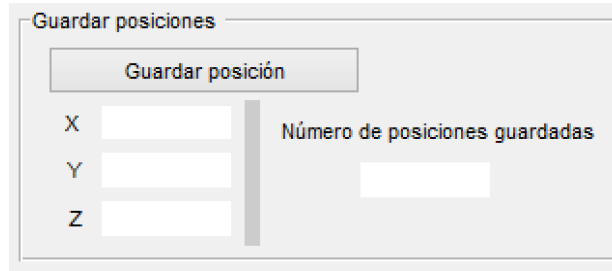
*Figura 8.6. Función botón para mover al robot en el eje X positivo*

Como se puede observar, aparecen dos funciones externas nuevas: **Configuracion\_Movimiento** y **Obtener\_Coordenadas\_Articulares**. La primera contiene los datos seleccionados de la configuración del movimiento actual (velocidad, precisión y tipo de movimiento) y la segunda sirve para recibir del servidor las coordenadas articulares de la posición actual del robot y actualizarlas. Esto es útil en caso de querer cambiar el tipo de movimiento de robot de coordenadas cartesianas a articulares. Así, el robot tendría los valores actualizados de las coordenadas articulares y se movería basándose en estas y no en las últimas actualizadas al pulsar un botón para mover una de esas coordenadas articulares.

### 8.1.3. Guardar posiciones.

Respecto al Escenario I, este panel es completamente nuevo. Su función consiste en guardar las posiciones que se deseen del robot para, después, reproducir la trayectoria de todos los puntos guardados y realizar la incisión en cualquier tipo de forma. Esto permite realizar incisiones con formas diferentes y en infinidad de direcciones, lo que supone un importante aumento de la flexibilidad de la aplicación.

En el panel, cuando se presiona el botón '*Guardar posición*', se almacena y se muestra por pantalla la posición de la herramienta del robot en ese momento, así como el número de posiciones guardadas (ver Figura 8.7).



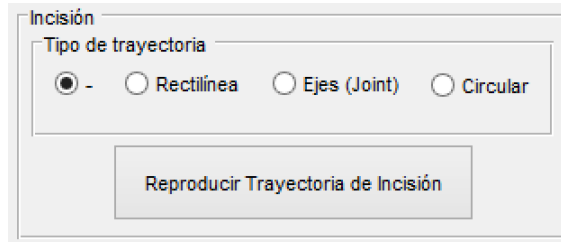
*Figura 8.7. Panel: Guardar posiciones*

Para almacenar todas las posiciones que se van guardando, se ha creado un vector de forma que cada vez que se presione el botón, se crea una nueva posición. Hay que destacar que para manejar posiciones se ha utilizado siempre el formato char, pero en este caso para almacenar las posiciones es necesario cambiar el formato a string con la función **convertCharsToStrings**.

Esto se debe a que para crear un vector de formato char, todos los char tienen que tener la misma longitud, es decir, el mismo número de caracteres. Como todas las posiciones no van a tener los mismos decimales, no van a tener el mismo número de caracteres, por lo que el programa lo interpretará como una matriz (teniendo cada fila el número de caracteres de cada posición) y aparecerán errores debido a que las dimensiones de la matriz no son correctas. Así, al realizar la conversión de char a string se consigue que todos los elementos contenidos en el string actúen como un único carácter, solventando el problema mencionado con anterioridad.

#### 8.1.4. Incisión.

Este último panel es muy simple y permite realizar la incisión. En él, simplemente se debe elegir si la trayectoria que va a seguir la herramienta para realizar la incisión va a ser rectilínea, según ejes (coordenadas articulares), o circular. Después, se presiona el botón '*Reproducir Trayectoria de Incisión*' para reproducir la trayectoria de todos los puntos guardados en el panel anterior (ver Figura 8.8).



*Figura 8.8. Panel: Incisión*

## 8.2. Operación de sutura.

Esta interfaz permite el control manual del robot de sutura y el control de la pinza encargada de colocar las grapas. Además, el robot de sutura en vez de seguir automáticamente la trayectoria seguida por el robot encargado de realizar la incisión y colocar el número de grapas elegido como en el Escenario I, se mueve manualmente hasta las posiciones en las que el cirujano crea conveniente poner grapas.

En la Figura 8.9 se muestra la interfaz de la operación de sutura.

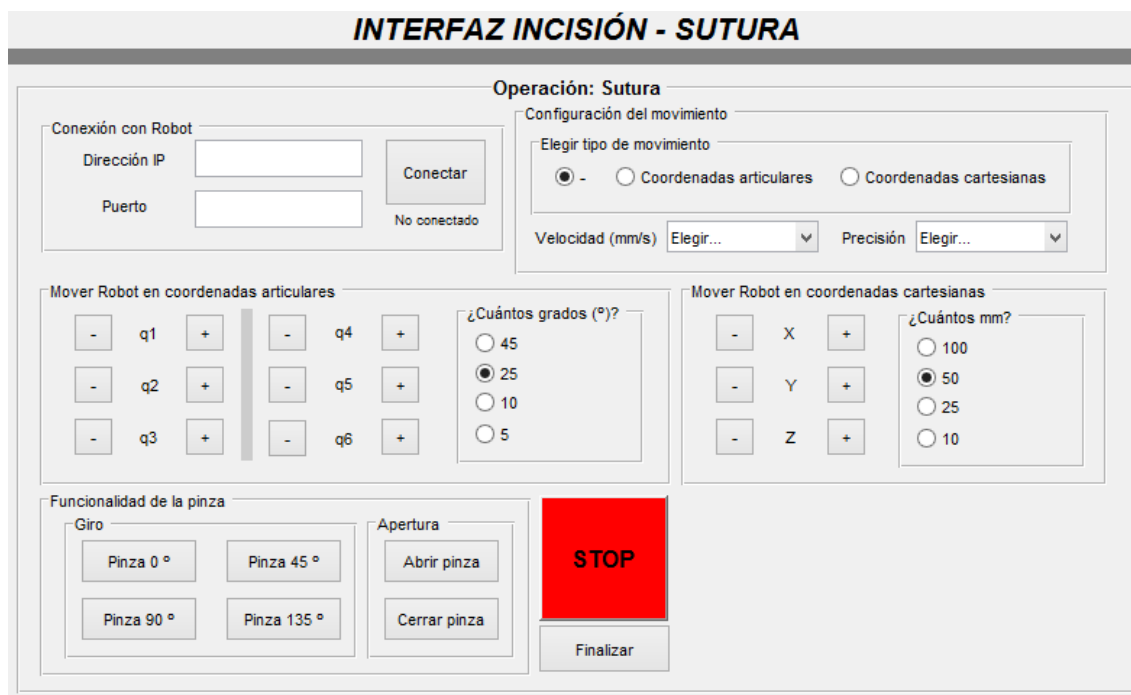


Figura 8.9. Escenario II: Interfaz gráfica de la operación de sutura

Como se puede observar, esta interfaz tiene bastantes elementos comunes con la interfaz de la operación de incisión de este mismo escenario. Por lo tanto, para evitar repeticiones, solo se va a explicar el panel 'Funcionalidad de la pinza', que es el que introduce una cierta novedad.

### 8.2.1. Funcionalidad de la pinza.

Este panel permite controlar el giro y la apertura de la herramienta. Como se ha comentado en el apartado 3.5, la pinza tiene 8 posiciones distintas: 4 posiciones diferentes de giro y para cada una de ellas la posibilidad de abrir o cerrar la pinza. Estas posiciones se pueden controlar con los botones que se muestran en la Figura 8.10.

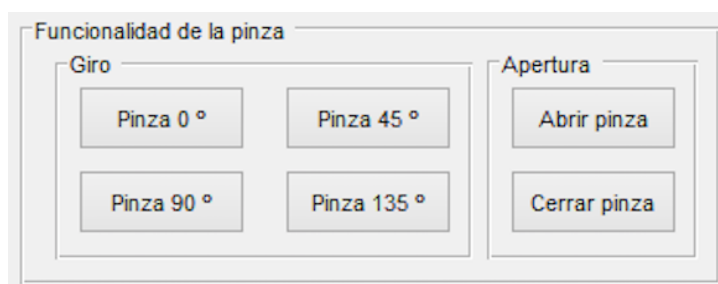


Figura 8.10. Panel: Funcionalidad de la pinza

Para conseguir que al presionar estos botones la pinza realice la función correspondiente, se creó en el controlador del robot de sutura 4 señales diferentes (**Digital Output**), denominadas: *AccionNormal*, *Accion45*, *Accion90* y *Accion135*. Para crearlas, se debe ir a la pestaña 'Controlador' → Configuración → I/O System. Estas señales se van a utilizar para enlazarlas con las señales de entrada creadas en el apartado 3.5 (para configurar la funcionalidad de la herramienta).

En *RobotStudio*, en la pestaña 'Simulación' → Lógica de estación aparecerán los dos controladores de los robots (con sus entradas y salidas) y la lógica de la funcionalidad de la pinza (apartado 3.5). Para enlazarlo, se debe unir cada señal de salida creada en el controlador del robot de sutura a la señal correspondiente en la lógica de la funcionalidad de la pinza (ver Figura 8.11).

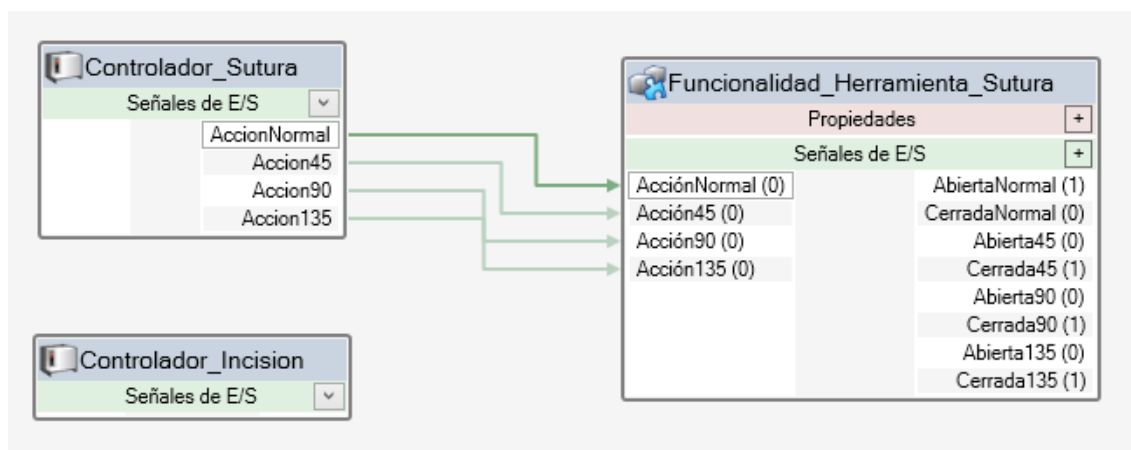


Figura 8.11. Lógica de estación Escenario II

Esto se realiza debido a que en *RAPID* solo se pueden activar y desactivar señales de salida propias de cada controlador. Entonces, al unir el controlador con la funcionalidad de la pinza, cuando se active la señal de salida de acción del controlador, ésta provocará que se active la propia de la funcionalidad de la herramienta y, por tanto, que gire o se abra/cierra.

Como se observa en la Figura 8.11, para conseguir que el valor de las señales de salida del componente inteligente de la herramienta cambie, se debe conseguir que la señal de entrada del mismo cambie, activando o desactivando la señal digital de salida del controlador de sutura para ello, pues éstas se encuentran enlazadas. Esto se puede conseguir mediante el uso de dos funciones diferentes:

**SETDO Sincronizar, 1; SETDO Sincronizar, 0;**  
**SET Sincronizar; RESET Sincronizar**

Con la primera opción (**SETDO**) se elige el valor que se quiere que alcance la señal, mientras que con la segunda opción la función **SET** sirve para activar la señal (1) y la función **RESET** sirve para desactivarla (0).

En la Figura 8.12 se puede observar un fragmento del programa RAPID en el que se muestra la programación de la funcionalidad de la herramienta a través de las señales creadas. Cabe destacar que cada vez que se presiona uno de los botones de giro del panel de la Figura 8.10, primero se envía un mensaje genérico al servidor que dice ‘Girar pinza’. Del mismo modo, cuando se presiona el botón de abrir o cerrar, primero se envía un mensaje genérico que dice ‘Apertura pinza’.

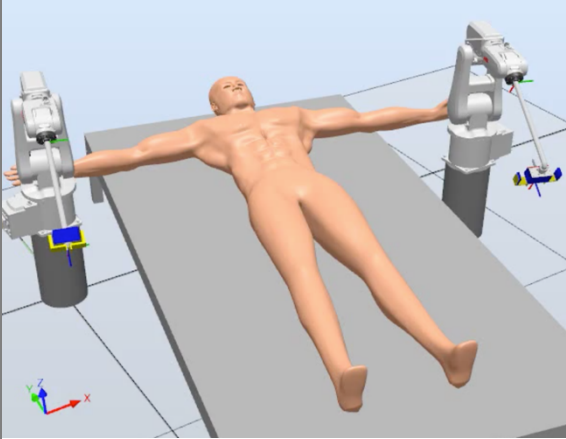
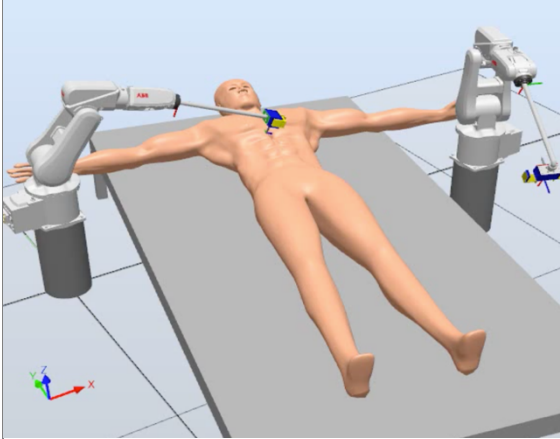
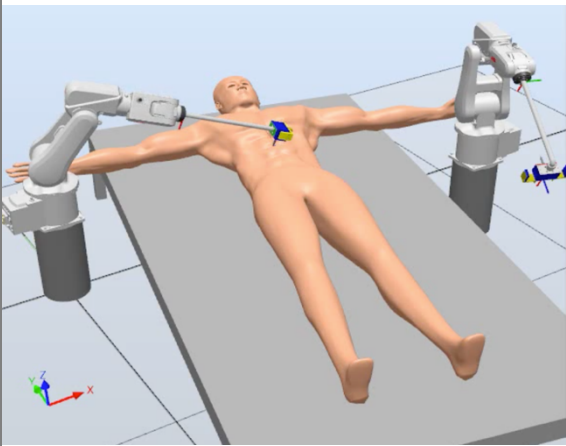
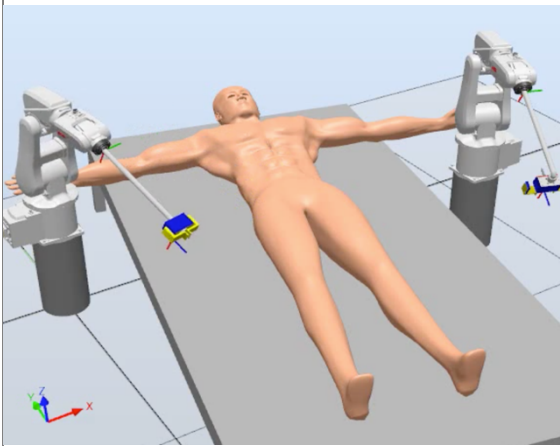
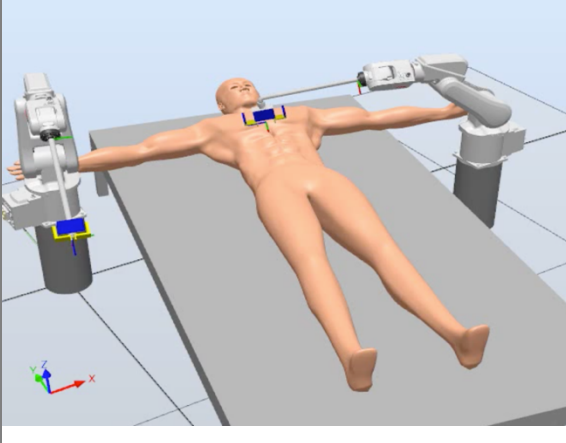
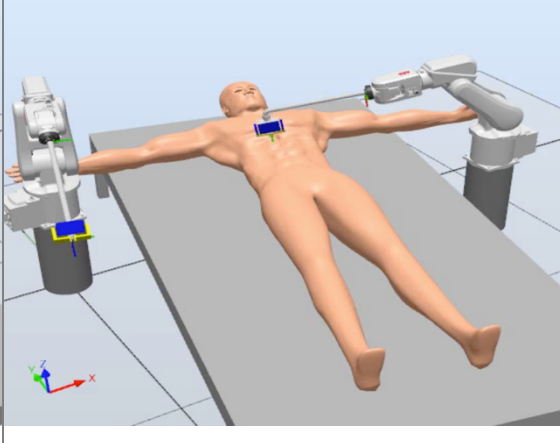
```

IF mensaje="Girar pinza" THEN
    SocketReceive cliente, \Str:=giro, \Time:=15;
    SETDO AccionNormal,1; Inicializamos los valores de las acciones
    SETDO Accion45,0;
    SETDO Accion90,0;
    SETDO Accion135,0;
    IF giro="Pinza 0" THEN
        SETDO AccionNormal,0;
    ELSEIF giro="Pinza 45" THEN
        SETDO Accion45,1;
    ELSEIF giro="Pinza 90" THEN
        SETDO Accion90,1;
    ELSEIF giro="Pinza 135" THEN
        SETDO Accion135,1;
    ENDIF

ELSEIF mensaje="Apertura pinza" THEN
    SocketReceive cliente, \Str:=apertura, \Time:=15;
    IF apertura="Abrir pinza" THEN
        IF giro="Pinza 0" THEN
            SETDO AccionNormal,0;
        ELSEIF giro="Pinza 45" THEN
            SETDO Accion45,1;
        ELSEIF giro="Pinza 90" THEN
            SETDO Accion90,1;
        ELSEIF giro="Pinza 135" THEN
            SETDO Accion135,1;
        ENDIF
    ELSEIF apertura="Cerrar pinza" THEN
        IF giro="Pinza 0" THEN
            SETDO AccionNormal,1;
        ELSEIF giro="Pinza 45" THEN
            SETDO Accion45,0;
        ELSEIF giro="Pinza 90" THEN
            SETDO Accion90,0;
        ELSEIF giro="Pinza 135" THEN
            SETDO Accion135,0;
        ENDIF
    
```

**Figura 8.12.** RAPID: Funcionalidad herramienta Escenario II

### 8.3. Esquema de las etapas de la simulación.

<p><b>1. Posición inicial</b></p> 	<p><b>2. Mover manualmente primer robot guardando los puntos que definen la trayectoria de la incisión</b></p> 
<p><b>3. Realización de incisión</b></p> 	<p><b>4. Primer robot vuelve a posición inicial</b></p> 
<p><b>5. Mover manualmente segundo robot a zona donde se quiere suturar</b></p> 	<p><b>6. Suturar en todos los puntos que se considere oportunos y volver a 1</b></p> 

*Tabla 8.1. Esquema etapas simulación Escenario II*



## CAPÍTULO 9. CONTROL SIMULTÁNEO DE OPERACIÓN DRENAJE-SUTURA.

En esta interfaz, la aplicación de estudio es la realización simultánea del drenaje de la sangre de la incisión y su momentánea sutura. Esta aplicación surgió de la idea de sincronizar ambos robots para realizar una tarea pero, continuar con la aplicación de los otros dos escenarios ya explicados era irrelevante dada la falta de semejanza con un caso real, como se explicó anteriormente

Cabe destacar que lo ideal hubiera sido poder implementar esta interfaz al Escenario II. En caso de haberlo hecho, habría que haber añadido un nuevo robot a la estación de *RobotStudio* y el motivo por el que se decidió no hacerlo fue que no se quería perder la coherencia con el resto del proyecto que está enfocado al número de robots y dimensiones de la estación disponible en el laboratorio de la Universidad.

A diferencia de los otros dos escenarios, en este caso no ha sido necesaria la creación de dos interfaces diferentes, una para cada tarea u operación. Además, contiene elementos mezclados de las interfaces de los otros dos escenarios, como la elección de la dirección de drenaje y sutura del Escenario I y el movimiento manual del robot del Escenario II (ver Figura 9.1).

**INTERFAZ DRENAJE - SUTURA**

Conexión con Robot Drenaje

Dirección IP

Puerto

No conectado

Conexión con Robot Sutura

Dirección IP

Puerto

No conectado

STOP

Configuración del movimiento

Elegir tipo de movimiento

☒ - ☐ Coordenadas articulares ☐ Coordenadas cartesianas

Velocidad (mm/s)

Precisión

Datos de la intervención

Longitud (mm)

Dirección

Número de grapas

Mover Robot en coordenadas articulares

-

q1

+

-

q4

+

-

q2

+

-

q5

+

-

q3

+

-

q6

+

¿Cuántos grados (°)?

☐ 45

☒ 25

☐ 10

☐ 5

Mover Robot en coordenadas cartesianas

-

X

+

-

Y

+

-

Z

+

¿Cuántos mm?

☐ 100

☒ 50

☐ 25

☐ 10

MOVER SINCRONIZADAMENTE

Figura 9.1. Escenario III: Interfaz gráfica de la operación drenaje-sutura



Como se hizo en el Escenario II, hay bastantes elementos comunes en la interfaz que ya se han explicado con anterioridad; por lo que para evitar repeticiones se explican a continuación únicamente los paneles que introducen algún tipo de novedad. Aunque no se comenten los paneles ‘Conexión con Robot’, cabe destacar que en este escenario se **pueden introducir los datos de los dos robots a la vez** para tener la conexión establecida desde el principio.

## 9.1. Datos de la intervención.

En este panel encontramos todos los elementos necesarios para introducir los datos de la intervención: la longitud de la incisión que hay que drenar, la dirección de la misma y el número de grapas que se desean colocar (ver Figura 9.2). Además, existe un botón que sirve para guardar la posición desde la que hay que empezar a drenar y así, poder enviársela a los controladores de los dos robots para que interpreten donde tienen que empezar a actuar.

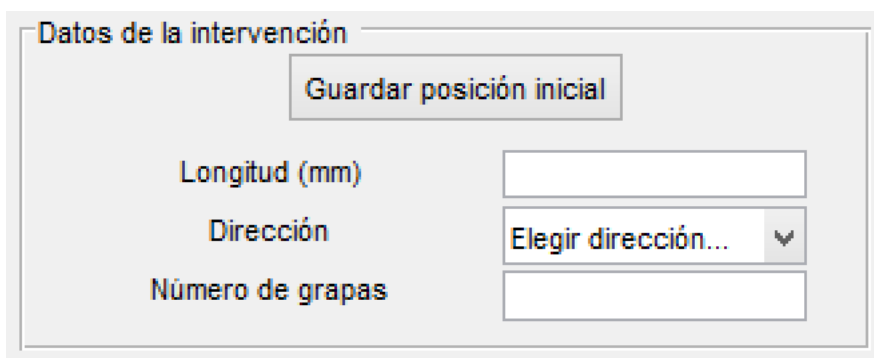
The image shows a software interface panel titled "Datos de la intervención". At the top right of the panel is a button labeled "Guardar posición inicial". Below this, there are three input fields. The first is labeled "Longitud (mm)" and is an empty text box. The second is labeled "Dirección" and is a dropdown menu with the text "Elegir dirección..." and a downward arrow. The third is labeled "Número de grapas" and is an empty text box.

Figura 9.2. Panel: Datos de la intervención

Cabe destacar que las posibles direcciones para realizar el drenaje de la incisión y su sutura no son las mismas que en el Escenario I. Esto se debe a que algunas de las direcciones que se podían elegir no podían utilizarse para realizar tareas simultáneas porque se producirían colisiones de las herramientas. Estas direcciones problemáticas eran: +X, 45° +X y 45° -Y. Por lo tanto, el **número de direcciones posibles** se reduce de 8 que eran posibles en el Escenario I, a 5 que son posibles en el escenario actual.

Asimismo, se debe recalcar que en este escenario ambos robots recorren la misma trayectoria siguiendo la misma dirección, pues de no ser así, se producirían colisiones entre las herramientas. Esto introduce una diferencia con respecto a los otros escenarios ya explicados, en los que no es obligatorio que se realice en el mismo sentido. En los otros escenarios, el sentido en el que se recorre la trayectoria no influye en las colisiones si no en la eficiencia de la aplicación.

## 9.2. Mover Sincronizadamente.

Este botón permite que los dos robots se muevan simultáneamente a lo largo de la trayectoria que define la incisión para realizar sus tareas correspondientes y en el mismo sentido. Para conseguir esta simultaneidad hay que crear nuevas señales de entrada y de salida en cada controlador para enlazarlas entre sí y crear tiempos de espera en función de si se activa o desactiva alguna de ellas.

En este caso se ha creado una nueva señal digital de salida (**Digital Output**) llamada *Sincronizar* en el controlador del robot de sutura. Además, se mantienen las señales de acción explicadas en el apartado 8.2.1, que permiten la apertura y el giro de la pinza. Asimismo, se ha creado una señal digital de entrada (**Digital Input**) en el controlador del robot de drenaje, llamada también *Sincronizar* (ver Figura 9.3).

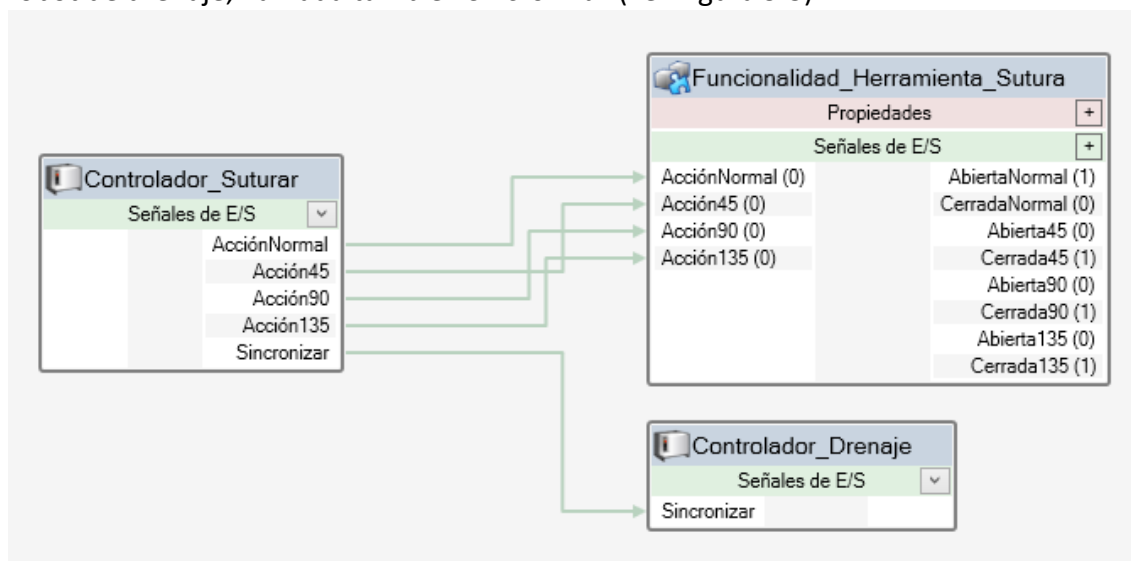


Figura 9.3. Lógica de estación Escenario III

Lo comentado previamente se ha realizado de esta manera debido a que el robot de drenaje se encuentra en la posición inicial para empezar a realizar la trayectoria porque el cliente lo ha movido con el control manual de la interfaz gráfica. Mientras tanto, el robot de sutura se encuentra en la posición de reposo lejos del cuerpo humano. Entonces, lo que se quiere conseguir es que los dos robots se muevan a la vez cuando los dos estén en la posición de inicio. Obviamente, no pueden estar los dos en la misma posición porque si no se producirían colisiones, por lo que el robot de sutura estará en una posición elevada 70 mm de la posición inicial en la que se encuentra el robot de drenaje.

La señal *Sincronizar* se emplea porque se ha utilizado en el programa del controlador de drenaje una función llamada **WaitDI**, cuya función es esperar a que la señal de entrada deseada tenga el valor deseado. Es decir, si se desea que el robot se mueva a partir del momento en el que la señal *Sincronizar* alcanza el valor 1, la sintaxis sería de siguiente:

**WaitDI Sincronizar, 1;**

Como se observa en la Figura 9.3, para conseguir que el valor de la señal de entrada *Sincronizar* del controlador del robot de drenaje cambie, hay que conseguir que la señal de salida *Sincronizar* del controlador del robot de sutura cambie también. Esto se consigue utilizando las dos funciones diferentes mencionadas en el apartado 8.2.1:

**SETDO Sincronizar, 1; SETDO Sincronizar, 0;**  
**SET Sincronizar; RESET Sincronizar**

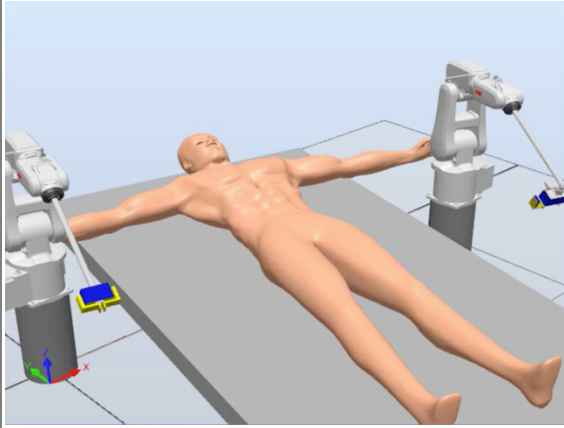
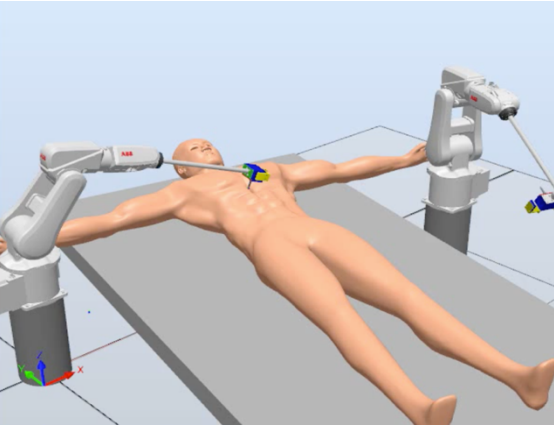
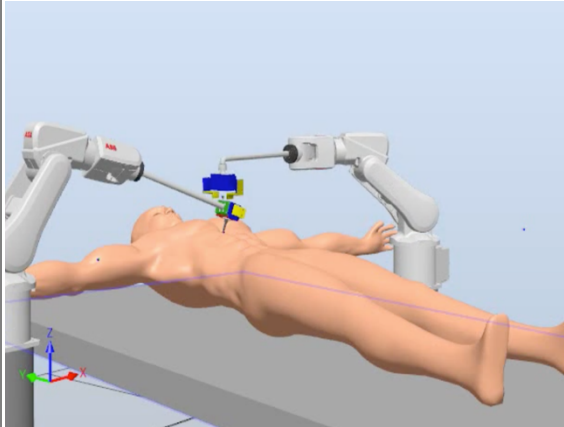
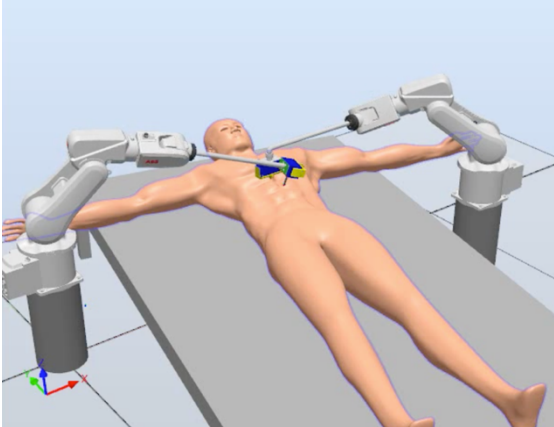
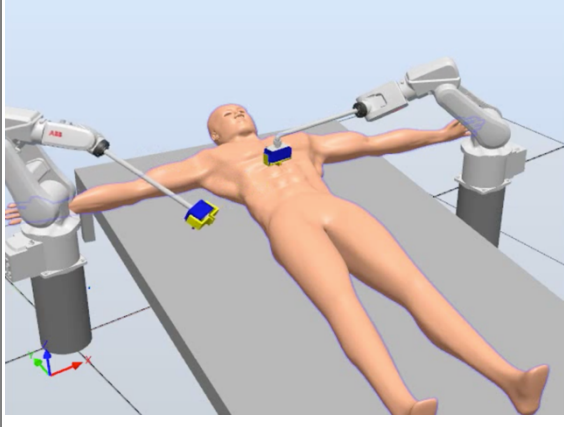
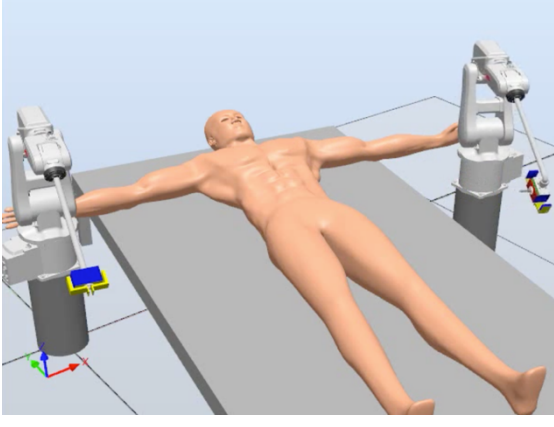
Finalmente, una vez se haya movido el robot de sutura a la posición en la que se va a iniciar la trayectoria, se activará la señal *Sincronizar* y, el robot de drenaje iniciará su movimiento simultáneamente al robot de sutura, aunque este último se moverá desfasado unos segundos en el tiempo para evitar problemas de colisiones.

También debe considerarse la funcionalidad de la herramienta, ya que el movimiento es automático y deberá girarse y abrirse o cerrarse en función de la posición en la que se encuentre. En este caso, el movimiento del robot de sutura es similar al que realiza en el Escenario I, por lo que su movimiento se reduce a **3·n posiciones**, donde n es el número de grapas que se desean colocar (apartado 7.2.3). Entonces, para girar y abrir o cerrar la pinza se ha generado un código en *RAPID* encabezado por un doble bucle **FOR**.

```
FOR i FROM 1 TO grapas DO
  FOR j FROM 1 TO 3 DO
    IF direccion="4" THEN !Direccion -X
      IF j=2 THEN
        SET AcciónNormal; !Cerrar pinza
        WaitTime 1;
        RESET AcciónNormal; !Abrir pinza
      ENDIF
    ELSEIF direccion="5" THEN !Direccion 45º -X
      IF j=1 THEN
        SET Acción45; !Girar pinza a 45º
      ELSEIF j=2 THEN
        RESET Acción45; !Cerrar pinza
        WaitTime 1;
        SET Acción45; !Abrir pinza
      ENDIF
    ELSEIF direccion="2" OR direccion="6" THEN !Direccion +Y, -Y
      IF j=1 THEN
        SET Acción90; !Girar pinza 90º
      ELSEIF j=2 THEN
        RESET Acción90; !Cerrar pinza
        WaitTime 1;
        SET Acción90; !Abrir pinza
      ENDIF
    ELSEIF direccion="3" THEN !Direccion 45º +Y
      IF j=1 THEN
        SET Acción135; !Girar pinza 135º
      ELSEIF j=2 THEN
        RESET Acción135; !Cerrar pinza
        WaitTime 1;
        SET Acción135; !Abrir pinza
      ENDIF
    ENDIF
  ENDIF
ENDIF
```

*Figura 9.4. RAPID: Funcionalidad de la herramienta Escenario III*

### 9.3. Esquema de las etapas de la simulación.

<p><b>1. Posición inicial</b></p> 	<p><b>2. Mover manualmente primer robot hasta posición de inicio de drenaje</b></p> 
<p><b>3. Mover automáticamente segundo robot a la posición del primer robot pero elevada 70 mm en el eje Z</b></p> 	<p><b>4. Movimiento sincronizado de drenaje y sutura</b></p> 
<p><b>5. Primer robot vuelve a posición inicial cuando termina de drenar mientras segundo robot termina de suturar</b></p> 	<p><b>6. Segundo robot vuelve a posición inicial cuando termina de suturar</b></p> 

*Tabla 9.1. Esquema etapas simulación Escenario III*

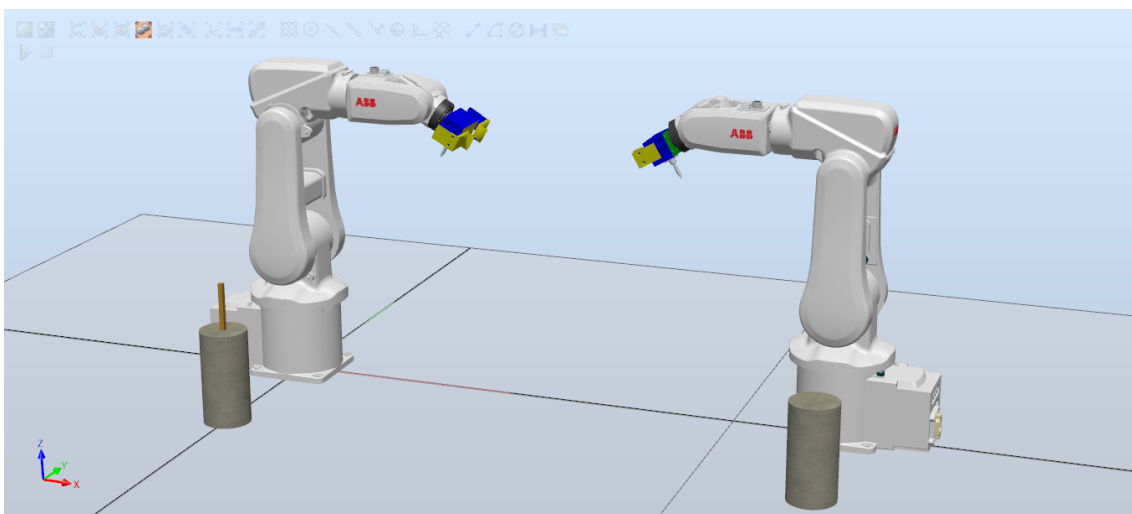
## CAPÍTULO 10. MANIPULACIÓN DE MATERIAL E INSTRUMENTAL QUIRÚRGICO.

Este último escenario es el que más cambios introduce con respecto a los anteriores y además, es el único que no posee una interfaz gráfica de usuario GUI. Se ha decidido no diseñarla pues simplemente se quiere demostrar la funcionalidad de la aplicación ya que es una tarea más específica, concreta y fija que las anteriores y en un futuro puede ser implementada en alguna de las interfaces explicadas con anterioridad.

Debido a la necesidad de utilizar diferente material e instrumental en una misma intervención quirúrgica, se ha estudiado es la manipulación de equipo quirúrgico entre dos robots usando movimientos coordinados y simultáneos. Para ello, ya que no se realizada ninguna operación sobre el paciente, se ha decidido cambiar la estación de trabajo para centrarse únicamente en los elementos importantes en este caso.

Cabe destacar que en la realidad este tipo de tareas se realizan siempre en posiciones alejadas del paciente que vaya a ser intervenido para garantizar su seguridad. Por lo tanto, se puede aislar esta parte de la estación de trabajo con respecto a la estación global, la cual sería el conjunto de esta estación de trabajo y la expuesta en los otros tres escenarios.

La estación de trabajo de este escenario consta de dos robots, cada uno con su respectiva herramienta, dos soportes para posar el material e instrumentar a manipular y el propio material (ver Figura 10.1). Es importante mencionar que las herramientas ancladas a la muñeca del robot son las mismas que en los otros escenarios, pero se han eliminado los mangos ya que los robots están situados a una distancia mucho menor el uno del otro. Además, solo se hace uso de la pinza de la herramienta para coger y soltar objetos.



*Figura 10.1. Estación de trabajo Escenario IV*

Así, debido a que las herramientas están libres de mangos, el componente inteligente que define la lógica de la funcionalidad de la pinza en *RobotStudio* se vuelve mucho mas sencillo. En este caso, dicha funcionalidad corresponde únicamente a la apertura y cierre de la pinza ya que el giro definido en otros escenarios lo hace por sí solo la muñeca del robot.

Para crear el componente inteligente se utilizan los mismos componentes que se han utilizado en los escenarios previos (*PoseMover*, *LogicGate NOT* y *LogicSRLatch*). También, se han creado señales de entrada y de salida para que la pinza se abra o cierre en función de los valores de estas señales. Se ha creado una señal de entrada, denominada *Acción*, y dos de salida: *Abierto* y *Cerrado*.

Entonces, cuando se active la señal *Acción*, se activará consecuentemente la señal de salida *Cerrado* y la pinza pasará a posición cerrada (1 cm); por el contrario, cuando ésta se desconecte, se activará la señal de salida *Abierto* y la pinza pasará a posición abierta (7 cm). El diseño de la lógica de este componente inteligente se muestra en la Figura 10.2.

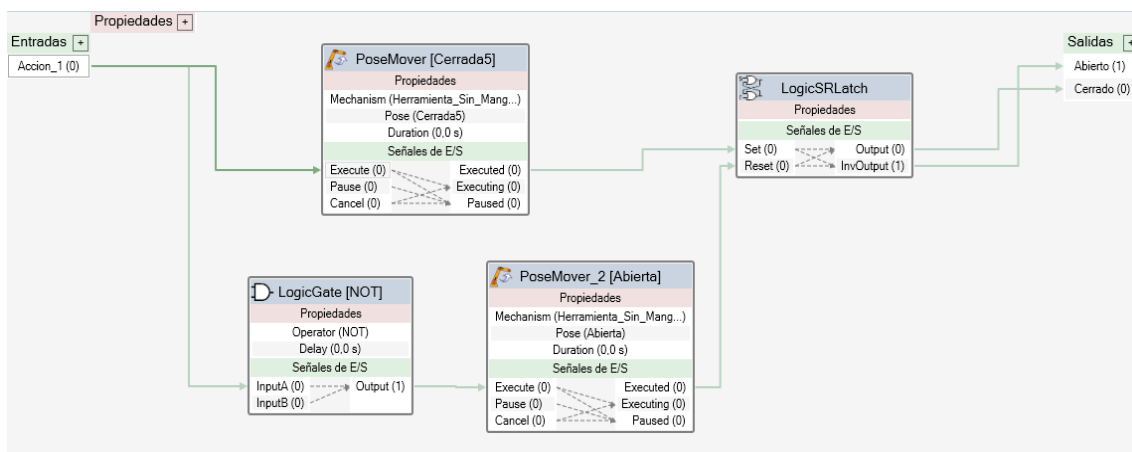


Figura 10.2. Componente inteligente herramienta Escenario IV

Este diseño de la lógica del componente inteligente de la herramienta se debe realizar de la misma forma con las herramientas de ambos robots.

Otro cambio a destacar con respecto a los otros escenarios es que en este se utiliza **un único controlador** para controlar ambos robots. Debido a la existencia de un único controlador se utiliza la función *MultiMove* con la que se consigue el movimiento coordinado y sincronizado de los dos robots en torno al objeto que se traspasa; es decir, en el momento de traspaso del objeto entre ambos robots, en vez de que uno se mueva totalmente hacia el otro y se lo entregue, ambos se acercarán a la vez entre sí simultáneamente para alcanzar una posición óptima para ambos.

Para utilizar esta función se debe ir en *RobotStudio* a la pestaña '*Posición Inicial*' → *Programación de trayectorias* → *MultiMove*.

Para configurar esta función primero se debe primero configurar el sistema, es decir, indicar qué robot lleva el objeto y cuál la herramienta. En segundo lugar, se debe configurar una trayectoria base formada por tres posiciones:

1. Posición inicial del robot que va a recibir el objeto.
2. Punto central de la base del objeto (zona por donde el robot va a agarrar el objeto), ya que por el otro extremo la tiene sujeta el robot que la va a entregar.
3. Posición inicial del robot que va a recibir el objeto.

Es posible que el programa muestre una advertencia indicando que alguna posición creada no es alcanzable por el robot y, por lo tanto, la trayectoria no se pueda realizar. Esto se debe a que el robot no puede alcanzar esa posición moviéndose únicamente él. Sin embargo, sí es alcanzable cuando se mueven coordinadamente ambos robots.

Por último, se debe seleccionar que robot debe moverse al otro para entregar el objeto, es decir, el robot que lleva el mismo. Esto sirve para que se cree una trayectoria similar a la del robot que recibe el objeto, pero usando su posición inicial y la posición en la que se desprende del objeto.

Cuando se han terminado de configurar estos parámetros, se debe presionar el botón *Reproducir*, que habilitará el comienzo de una simulación en la que ambos robots se mueven de forma coordinada pasando por las posiciones mencionadas previamente.

Una vez la simulación haya finalizado, se activará una nueva pestaña dentro de la función *MultiMove* llamada '*Crear trayectorias*'. Cuando se presiona ese botón, el programa crea automáticamente para cada uno de los robots una trayectoria que contiene las posiciones que deben alcanzar ambos para realizar el movimiento coordinado.

El programa principal consta de 3 trayectorias:

1. El primer robot se acerca al objeto, cierra la pinza, y la mueve con él hasta la posición inicial del robot.
2. Ambos robots se mueven coordinadamente hasta la posición en la que el segundo robot puede agarrar la base del objeto. En ese momento, el segundo robot cierra la pinza para agarrar el objeto, mientras que el primer robot abre su pinza para soltarlo. Tras esto, cada robot vuelve a su posición inicial. Estas trayectorias son las que se han creado pulsando el botón '*Crear trayectorias*' de la función *MultiMove*.
3. El segundo robot mueve el objeto hasta el soporte, donde abre la pinza para soltarlo y vuelve a la posición inicial.

Para conseguir que el segundo robot espere hasta que el primer robot haya movido el objeto hasta su posición inicial, se hace uso de unas variables que definen la tarea de cada robot y el punto de sincronización.

```
PERS tasks Sincronizar{2}:=[["T_ROB1"],["T_ROB2"]];  
VAR syncident sincro;
```

Después, en el momento en el que se desee comenzar el movimiento coordinado, se añade en la línea del código que corresponda la siguiente función:

```
WaitSyncTask sincro, Sincronizar;
```



Esta función hace que cuando uno de los programas de los robots llegue a esa línea del código, espere hasta que el programa del otro robot también alcance esa línea en su correspondiente programa, consiguiendo así la sincronización.

Finalmente, para conseguir que el robot abra y cierre la pinza y agarre o suelte el objeto se debe diseñar la lógica de la estación (ver Figura 10.3). Para ello, se deben crear dos señales digitales de salida del controlador (Digital Output), una para cada componente inteligente de las herramientas, y se añaden dos componentes nuevos: *Attacher* y *Detacher*. El primero sirve para conectar dos objetos (una de las dos herramientas y el objeto a mover), es decir, los dos objetos seleccionados se moverán a la vez; el segundo sirve para desconectar esos objetos.

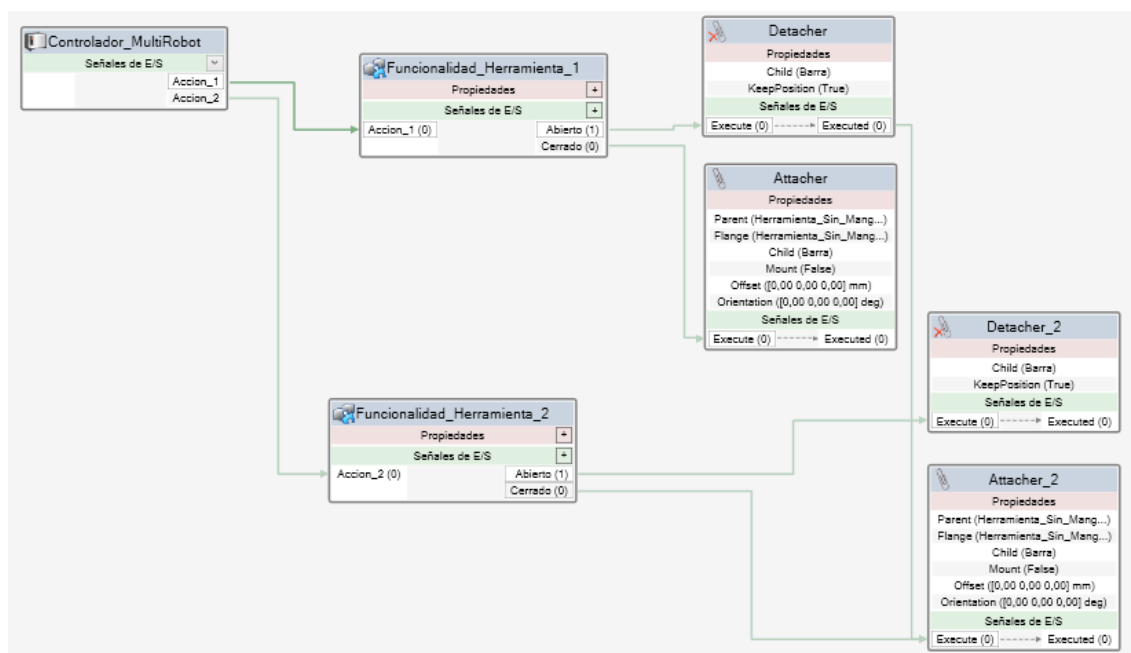


Figura 10.3. Lógica de estación Escenario IV

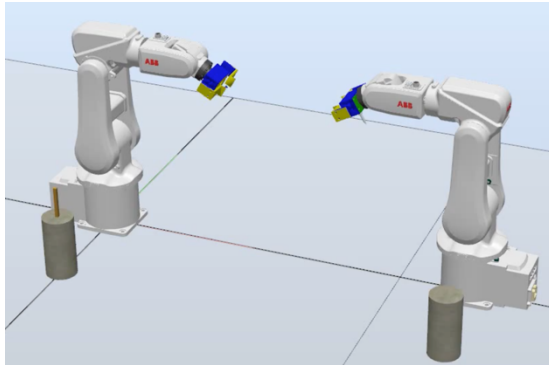
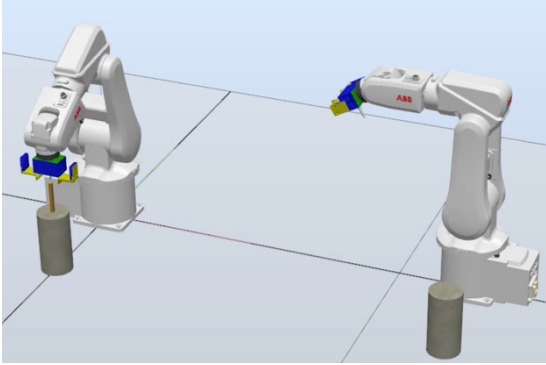
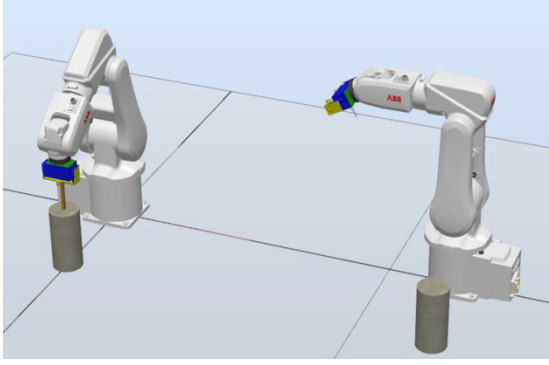
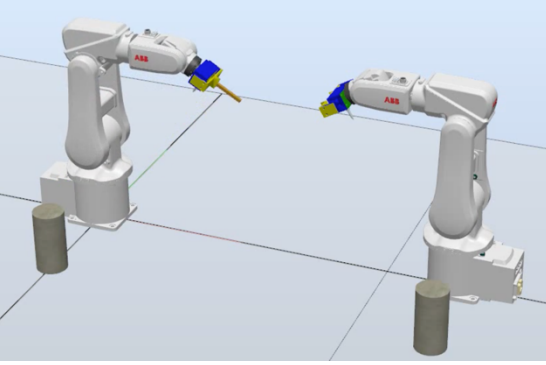
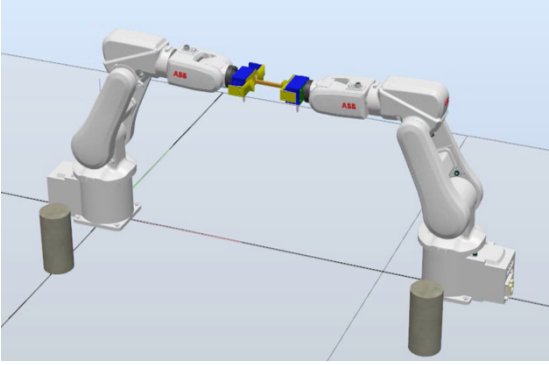
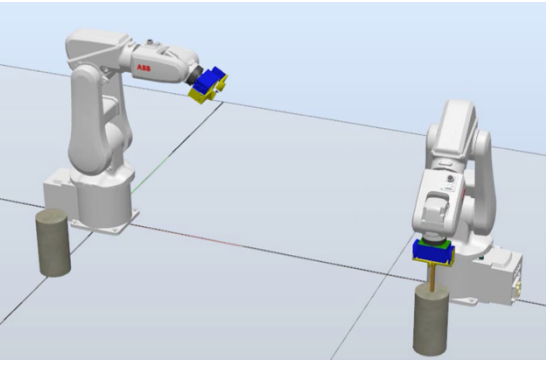
Debido a los enlaces mostrados en la Figura 10.3, cuando se active la señal digital de salida *Accion\_1*, la pinza se cerrará y la herramienta del primer robot se conectará al objeto, moviéndose simultáneamente. Cuando se desactive la señal, la pinza de esa herramienta se abrirá y el objeto y la herramienta se desconectarán. Lo mismo ocurre con la señal *Accion\_2*.

Cabe destacar que se han enlazado el componente *Detacher* del primer robot y el componente *Attacher* del segundo. Esto se debe a que si se conecta el segundo robot al objeto sin que el primer robot se haya desconectado pueden aparecer errores ya que el objeto se tendría que mover simultáneamente a los dos robots, **lo cual sería imposible porque los dos robots se mueven en diferentes trayectorias**. Para evitar este tipo de errores, se han enlazado los componentes mencionados previamente de forma que hasta que el robot que está conectado al objeto no se desconecte, no se permita a otro robot conectarse al mismo objeto.



Esto solo tiene relevancia en el ámbito de la simulación, pues en su aplicación real cuando la pinza del robot se cierra, por la presión que ejerce sobre el objeto, éste se mueve con ella. En cambio, en la simulación, como se ha mencionado anteriormente, se debe especificar que cuando se cierra la pinza de la herramienta, el objeto debe moverse con ella.

Finalmente, para poder comprender totalmente cómo funciona esta aplicación, en la Tabla 10.1 se puede observar un esquema en el que se detallan las distintas etapas que engloba la aplicación.

<p><b>1. Posición inicial</b></p> 	<p><b>2. Primer robot se mueve hasta posición para coger objeto</b></p> 
<p><b>3. Primer robot cierra la pinza y agarra el objeto</b></p> 	<p><b>4. Primer robot se mueve hasta posición inicial con el objeto agarrado</b></p> 
<p><b>5. Ambos robots se mueven coordinadamente para intercambiarse la pieza (MultiMove)</b></p> 	<p><b>6. Segundo robot con la pieza se mueve hasta la posición donde debe dejarla, la suelta y vuelve a 1</b></p> 

*Tabla 10.1. Esquema etapas simulación Escenario IV*

# CAPÍTULO 11. CONCLUSIONES Y LÍNEAS FUTURAS.

---

Tal y como se expone en los objetivos del proyecto, el presente Trabajo Fin de Grado se ha enfocado a la estación de trabajo del laboratorio de la Universidad para poder evaluar una prueba real de las aplicaciones de estudio. Debido a la crisis sanitaria del COVID-19 y el cambio de la docencia a no presencial, se ha imposibilitado el acceso a los laboratorios de la Universidad y, por consiguiente, probar las aplicaciones creadas en la vida real. Por eso mismo, para completar el proyecto se ha ampliado el número de aplicaciones de estudio de una inicial a 4. Se espera que en un futuro todas las aplicaciones mostradas en el presente proyecto puedan ser probadas en la realidad.

El principal problema que se ha detectado en la mayoría de aplicaciones es que el tiempo de ejecución es mayor que si se realiza por métodos convencionales dada la agilidad que requiere el correcto uso de la interfaz para mover al robot manualmente hasta las posiciones de inicio. El tiempo estimado en la simulación es alrededor de 3-4 minutos para realizar el posicionamiento, la incisión y la sutura. En cambio, por el método convencional en el que el cirujano es el encargado de realizar la incisión, el acto de situar la herramienta en la posición de inicio es prácticamente instantáneo.

Sin embargo, la precisión de este tipo de cirugías es mucho mayor que cuando la realiza un profesional médico, eliminando las posibles imprecisiones que conlleva la implicación de un ser humano (inestabilidad por un pulso tembloroso, ). Como consecuencia de esta precisión, las incisiones se realizan de una manera mucho más limpia y, por lo tanto, su cicatrización es más rápida. Este hecho es muy importante ya que lo uno de los aspectos que se busca con este tipo de cirugía es una mejor recuperación.

Otro problema que se ha encontrado es el diseño de las herramientas. La herramienta tiene unos mangos de gran longitud para que ambos robots puedan alcanzar puntos en común y realizar las acciones sobre ellos. No obstante, tal longitud trae consigo una serie de problemas o desventajas. En primer lugar, imposibilita que la herramienta del robot alcance determinadas posiciones. Además, para alcanzar algunas posiciones puede realizar movimientos poco orgánicos como giros muy cerca del cuerpo del paciente que pueden poner en riesgo su salud.

Para solventar el problema del diseño de las herramientas se plantea realizar un rediseño de la herramienta de forma que sea más ergonómica. Esto se debe a que la herramienta real está disponible en el laboratorio de la Universidad y tiene un peso considerable. Además, en algunas pruebas que se hicieron con ella, el peso de la misma hacía que algunos movimientos del robot no fueran tan precisos como deberían ser y, si se añade la longitud del mango diseñado en este proyecto con su correspondiente peso, este error se puede agravar. Asimismo, se debe tener en cuenta el centro de masas y el momento de inercia de la herramienta, que en la simulación no ha sido necesario tener en cuenta y que afecta notablemente a la aplicación en la vida real. Por lo tanto, con tal

de que la precisión del robot no se viera afectada, una posible mejora a considerar sería cambiar las dimensiones y materiales de la herramienta para hacerla más pequeña y ligera, así como cambiar la forma y longitud de los mangos para conseguir que pueda alcanzar más posiciones sin errores.

En base al problema del elevado tiempo de ejecución, se plantea como línea futura para la continuación del proyecto el uso de inteligencia artificial (IA), especialmente la visión por computador. Es decir, se podría incorporar al robot una cámara para que tome fotos del paciente, las procese mediante un programa que identifique diferentes colores y siga la trayectoria que el cirujano ha dibujado con un rotulador de color sobre el cuerpo del paciente. Con esta solución, ahorraríamos un tiempo considerablemente alto en el posicionamiento del robot.

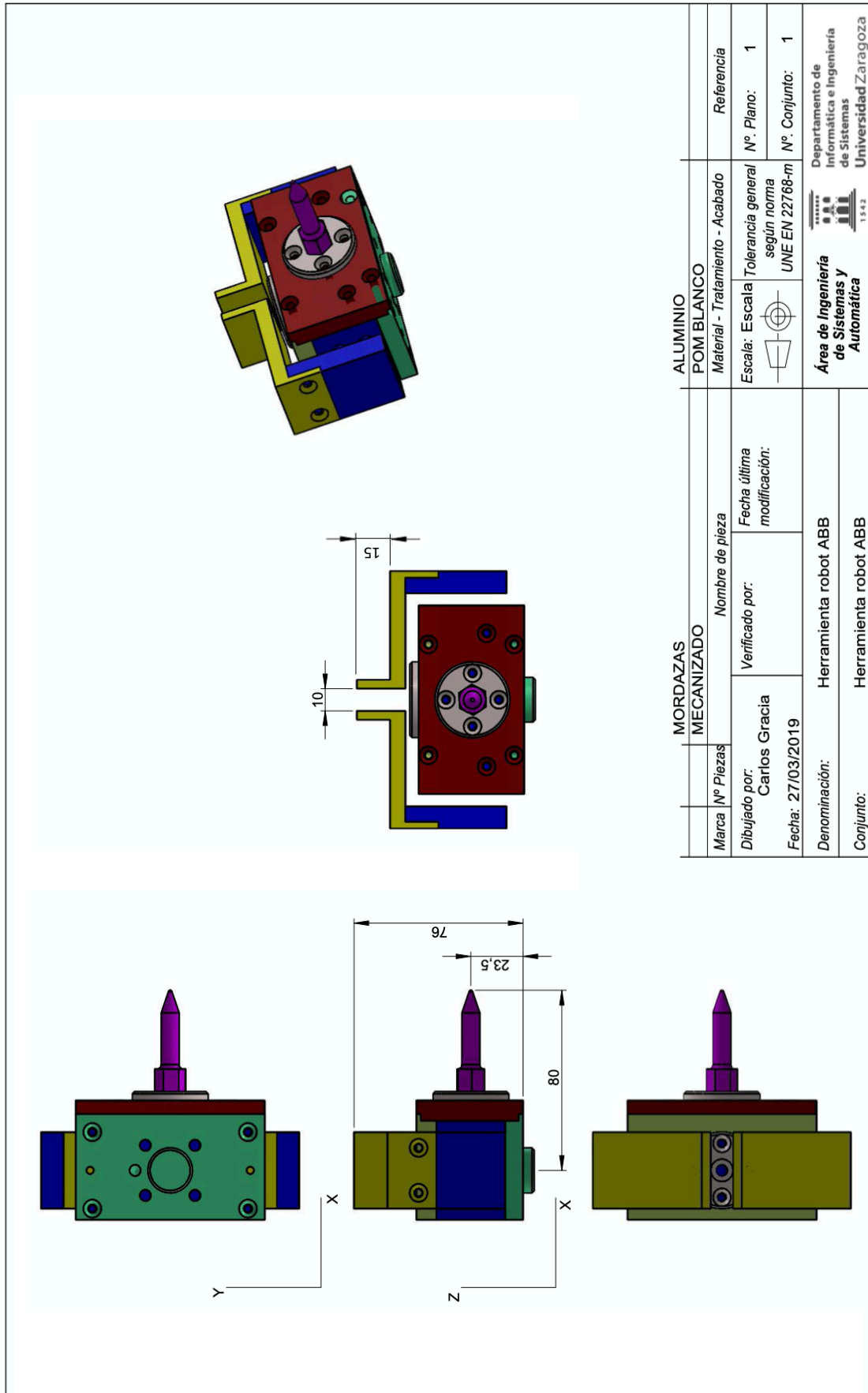
Si se abordara el tema de inteligencia artificial se estarían enlazando dos de los ámbitos más prominentes en lo que a avances tecnológicos se refiere. En mi opinión, dicha unión lleva consigo una potencialidad inconmensurable que a día de hoy es bastante difícil de comprender ya que a día de hoy solo existen prototipos en desarrollo que entrelazan ambas disciplinas. Por lo tanto, considero que aplicando conocimientos de inteligencia artificial al presente proyecto, éste alcanzaría una potencialidad, eficacia y reconocimiento mucho mayor.

## BIBLIOGRAFÍA

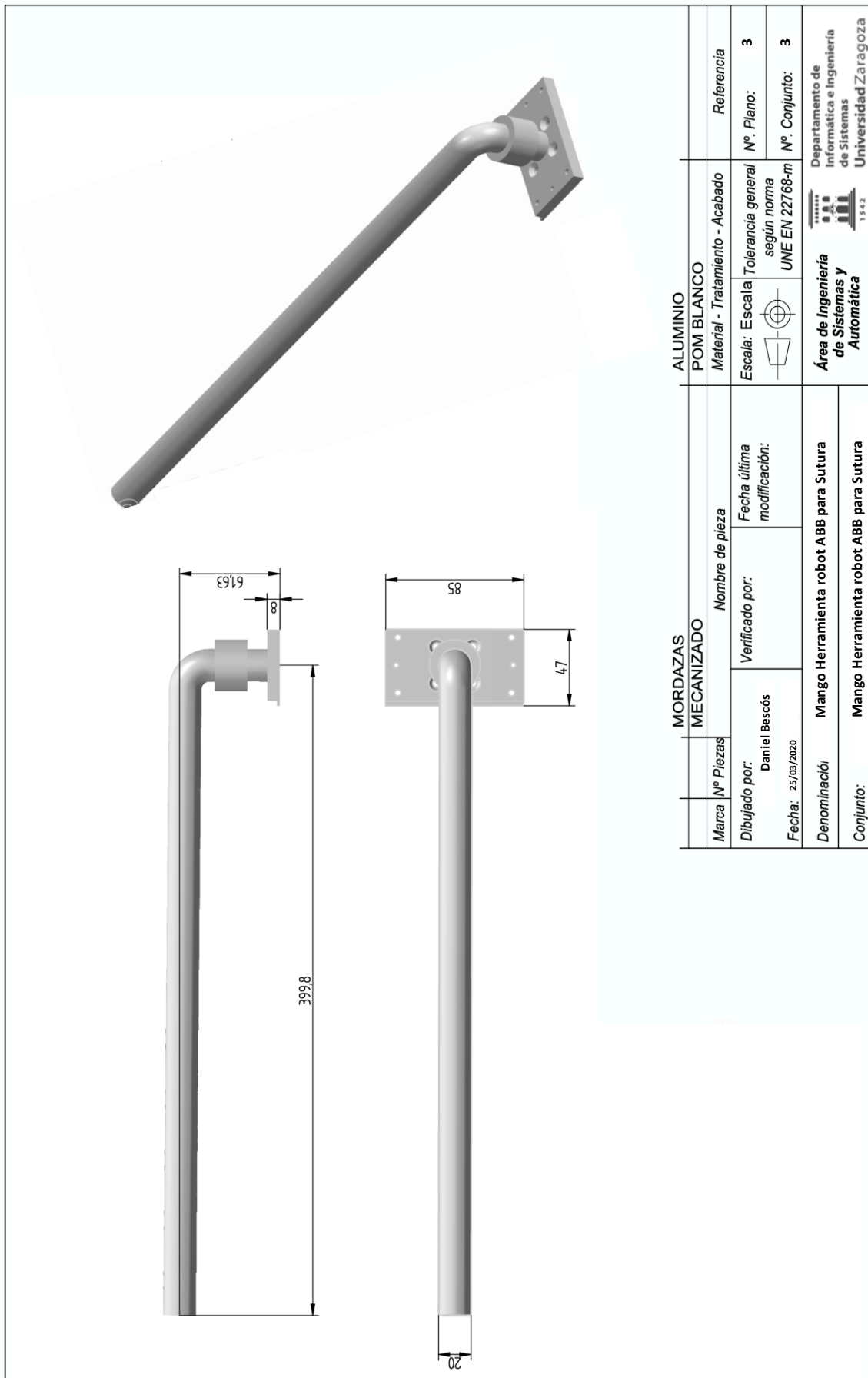
- [1] Renta García, E. (2019). "Desarrollo de un sistema plóter basado en un robot ABB IRB120". Trabajo Fin de Grado. Universidad de Zaragoza.
- [2] Ramos, L. (18 de Noviembre de 2016). "Pronto habrá ingenieros trabajando codo con codo con equipos clínicos". *El Comercio*.  
Recuperado de <https://www.elcomercio.es/gijon/201611/18/pronto-habra-ingenieros-trabajando-20161118001928-v.html>
- [3] Prieto, M. (5 de Mayo de 2016). "Nanotecnología, exoesqueletos, impresión 3D: Así es la medicina del futuro". *Expansión*.  
Recuperado de <https://www.expansion.com/economia-digital/innovacion/2016/05/05/572b63a222601d67638b4590.html>
- [4] Barrientos, A., Peñín, L. F., Balaguer, C. & Aracil, R. (1997). "*Fundamentos de robótica*". Madrid, España: McGraw-Hill.
- [5] Asimov, I. (1950). "*I, Robot*". Boston, Estados Unidos: Gnome Press.
- [6] Sánchez-Martín, F.M., Jiménez Schlegl, P., Millán Rodríguez, F., Salvador-Bayarri, J., Monllau Font, V., Palou Redorta, J., & Villavicencio Mavrich, H.. (2007). "Historia de la robótica: de Arquitas de Tarento al Robot da Vinci (Parte II)". *Actas Urológicas Españolas*, 31(3), 185-196.
- [7] Bonev, I. (24 de Enero de 2003). "The True Origins of Parallel Robots". *ParalleMIC*.  
Recuperado de <https://www.parallemic.org/Reviews/Review007.html>
- [8] Malone, B. (26 de Septiembre de 2011). "George Devol: A Life Devoted to Invention, and Robots". *IEEE Spectrum*.  
Recuperado de <https://spectrum.ieee.org/automaton/robotics/industrial-robots/george-devol-a-life-devoted-to-invention-and-robots>
- [9] "¿Conoces la historia de los robots colaborativos?". (11 de Abril de 2019). Grupo Kopar, Automatización y Robótica Industrial.  
Recuperado de <https://www.kopar.com.mx/blog/2019/04/11/conoces-la-historia-de-los-robots-colaborativos/>
- [10] Pelegrí, J. (21 de Junio de 2019). "Beneficios de la automatización con robots colaborativos". Universal Robots.  
Recuperado de <https://blog.universal-robots.com/es/beneficios-robots-colaborativos>
- [11] Kwoh, Y., Hou, J., Jonckheere, E. & Hayati, S. (1988). "A robot with improved absolute positioning accuracy for CT guided stereotactic brain surgery". *IEEE Trans Biomed Eng*, 35:153-61

- [12] B. Samadi, D. "Orígenes de la cirugía robótica". *Robotic Oncology*. Recuperado de <https://www.roboticoncology-es.com/cirugia-robotica-origenes/>
- [13] Pruthi, R. & Wallen, E. (2009). "Current Status of Robotic Prostatectomy: Promises Fulfilled". *J Urol*, 181:2420-3.
- [14] Garcia Berro, M. (Noviembre de 2004). "El Futuro de la Cirugía Mínimamente Invasiva". Recuperado de [http://panelfenin.es/uploads/fenin/documento\\_estudios/pdf\\_documento\\_18.pdf](http://panelfenin.es/uploads/fenin/documento_estudios/pdf_documento_18.pdf)
- [15] "¿Qué es el robot Da Vinci?". ABEX Excelencia Robótica. Recuperado de <http://www.abexsl.es/es/robot-da-vinci/que-es>
- [16] Marínez-Ramos, C. (2007). "Cirugía robótica (I): origen y evolución". *Asecma*. Cir May Amb 2007 (12), 89-96.
- [17] Eveleth, R. (27 de Mayo de 2014). "Los cirujanos que operan a cientos de kilómetros de distancia". *BBC*. Recuperado de [https://www.bbc.com/mundo/noticias/2014/05/140520\\_vert\\_fut\\_salud\\_cirujano\\_a\\_distancia\\_gtg](https://www.bbc.com/mundo/noticias/2014/05/140520_vert_fut_salud_cirujano_a_distancia_gtg)
- [18] "Los beneficios del 5G llegan a la robótica". (27 de Febrero de 2019). *Network World*. Recuperado de <https://www.networkworld.es/convergencia/los-beneficios-del-5g-llegan-a-la-robotica#>
- [19] "Manual de referencia técnica: Instrucciones, funciones y tipos de datos de RAPID". (2018). *ABB Automation Technology Products AB*. Recuperado de <https://abb.sluzba.cz/Pages/Public/IRC5UserDocumentationRW6/es/3HAC050917%20TRM%20RAPID%20RW%206-es.pdf>
- [20] Denavit, J. & Hartenberg, R.S. (Junio 1995). "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices". *Journal of Applied Mechanics*. Recuperado de <https://home.konkuk.ac.kr/~cgkang/courses/robotics/courseMaterials/DHpaper.pdf>
- [21] Guerrero, J.J. "Apuntes de la asignatura Automatización Flexible y Robótica". *Universidad de Zaragoza*.
- [22] "Especificaciones del producto: IRB 120". (2019). *ABB Automation Technology Products AB*. Recuperado de <https://library.e.abb.com/public/8c35b9eababc442b926c6728983f0898/3HAC035960%20PS%20IRB%20120-es.pdf?x-sign=qGe1B0hbjWSHTZJYglTp17x8aplwTn9mt6MGSATVwjSO5Q3j5exd3suE4N5p7xtN>

# ANEXO I. PLANOS DE LA HERRAMIENTA.









## ANEXO II. MANUAL DE USUARIO.

---

En este manual se va a explicar la organización y el modo de ejecución de los diferentes programas y sistemas que componen las aplicaciones de estudio. Las versiones de los programas utilizados para la programación han sido *RobotStudio 2019.2* con *RobotWare 6.10* y *MATLAB R2017b*, ambos para Windows 8 (64 bits).

Los archivos necesarios para la ejecución de las aplicaciones están ordenados en diferentes carpetas. La carpeta principal, cuyo nombre es el propio del presente Trabajo Fin de Grado, está dividida en cuatro carpetas, una para cada escenario.

Dentro de cada una de estas carpetas aparece una carpeta en la que están guardados los archivos *MATLAB* y otra en la que están guardados los archivos *RAPID*. En la carpeta de los archivos *MATLAB* se encuentra tanto el archivo principal de la interfaz gráfica de usuario GUI como el resto de funciones auxiliares necesarias para su correcto funcionamiento. Los archivos *RAPID* se han guardado en carpetas para evitar que si el archivo de *RobotStudio* de la aplicación no se puede abrir o existe algún otro problema, no se pierda todo el trabajo realizado.

Además de las carpetas de los archivos de *MATLAB* y *RAPID*, también se encuentra el video de la simulación y el archivo **Pack&Go** del escenario correspondiente. En este archivo **Pack&Go** se encuentran todos los elementos importados, controladores virtuales y módulos de programación necesarios para el correcto funcionamiento de la aplicación.

Finalmente, para ejecutar los programas hay que seguir los siguientes pasos:

1. Abrir el archivo **Pack&Go** (formato \*.rspag) del escenario que se desee.
2. Abrir *MATLAB* y escoger como directorio la carpeta de archivos *MATLAB* del escenario correspondiente.
3. Una vez abierto el archivo **Pack&Go** de *RobotStudio*, en la pestaña 'Simulación' presionar el botón **Reproducir**.
4. En el directorio abierto en *MATLAB*, ejecutar el archivo:
  - a. **Interfaz\_Incision.m** si se desea ejecutar el Escenario I y II. Aunque tengan el mismo nombre, la programación es diferente y por lo tanto cada archivo se encuentra en la carpeta de archivos *MATLAB* de su correspondiente escenario.
  - b. **Interfaz\_Drenaje\_Sutura.m** si se desea ejecutar el Escenario III.
  - c. **Manipulacion\_Material\_Instrumental\_Quirurgico.m** si se desea ejecutar el Escenario IV.
5. Finalmente, en el caso de que se haya ejecutado un escenario que tenga interfaz gráfica de usuario GUI, se deberán introducir los datos según lo explicado en los capítulos 7, 8 y 9.

## ANEXO III. SCRIPTS DE MATLAB.

### Función: Modelo Geométrico Directo.

```
function [handles] = Modelo_Geometrico_Directo(hObject)

handles = guidata(hObject);

%Eslabón 1
d=290;
alpha=-90;
a=0;
A1=[cosd(handles.q1), -sind(handles.q1)*cosd(alpha), sind(handles.q1)*sind(alpha),
a*cosd(handles.q1);
sind(handles.q1), cosd(alpha)*cosd(handles.q1), -cosd(handles.q1)*sind(alpha),
a*sind(handles.q1); 0, sind(alpha), cosd(alpha), d; 0, 0, 0, 1];

%Eslabón 2
d=0;
alpha=0;
a=270;
A2=[cosd(handles.q2), -sind(handles.q2)*cosd(alpha), sind(handles.q2)*sind(alpha),
a*cosd(handles.q2);
sind(handles.q2), cosd(alpha)*cosd(handles.q2), -cosd(handles.q2)*sind(alpha),
a*sind(handles.q2); 0, sind(alpha), cosd(alpha), d; 0, 0, 0, 1];

%Eslabón 3
d=0;
alpha=-90;
a=70;
A3=[cosd(handles.q3), -sind(handles.q3)*cosd(alpha), sind(handles.q3)*sind(alpha),
a*cosd(handles.q3);
sind(handles.q3), cosd(alpha)*cosd(handles.q3), -cosd(handles.q3)*sind(alpha),
a*sind(handles.q3); 0, sind(alpha), cosd(alpha), d; 0, 0, 0, 1];

%Eslabón 4
d=302;
alpha=90;
a=0;
A4=[cosd(handles.q4), -sind(handles.q4)*cosd(alpha), sind(handles.q4)*sind(alpha),
a*cosd(handles.q4);
sind(handles.q4), cosd(alpha)*cosd(handles.q4), -cosd(handles.q4)*sind(alpha),
a*sind(handles.q4); 0, sind(alpha), cosd(alpha), d; 0, 0, 0, 1];

%Eslabón 5
d=0;
alpha= -90;
a=0;
A5=[cosd(handles.q5), -sind(handles.q5)*cosd(alpha), sind(handles.q5)*sind(alpha),
a*cosd(handles.q5);
sind(handles.q5), cosd(alpha)*cosd(handles.q5), -cosd(handles.q5)*sind(alpha),
a*sind(handles.q5); 0, sind(alpha), cosd(alpha), d; 0, 0, 0, 1];

%Eslabón 6
d=72;
alpha=0;
a=0;
A6=[cosd(handles.q6), -sind(handles.q6)*cosd(alpha), sind(handles.q6)*sind(alpha),
a*cosd(handles.q6);
sind(handles.q6), cosd(alpha)*cosd(handles.q6), -cosd(handles.q6)*sind(alpha),
a*sind(handles.q6); 0, sind(alpha), cosd(alpha), d; 0, 0, 0, 1];
```



```
%Matriz de Transformación Homogenea
T1=Multiplicar_Matrices(A1,A2);
T2=Multiplicar_Matrices(T1,A3);
T3=Multiplicar_Matrices(T2,A4);
T4=Multiplicar_Matrices(T3,A5);
T=Multiplicar_Matrices(T4,A6);

%Coordenadas [X,Y,Z] y cuaternios.
global coordenada_x coordenada_y coordenada_z;
coordenada_x=T(1,4);
coordenada_y=T(2,4);
coordenada_z=T(3,4);

X=char(num2str(T(1,4)));
Y=char(num2str(T(2,4)));
Z=char(num2str(T(3,4)));

posicion = [X, ',', Y, ',', Z];

cuaternio1=char(num2str(0.5*sqrt(T(1,1)+T(2,2)+T(3,3)+1)));
cuaternio2=char(num2str(sign(T(3,2)-T(2,3))*0.5*sqrt(T(1,1)-T(2,2)-T(3,3)+1)));
cuaternio3=char(num2str(sign(T(1,3)-T(3,1))*0.5*sqrt(-T(1,1)+T(2,2)-T(3,3)+1)));
cuaternio4=char(num2str(sign(T(2,1)-T(1,2))*0.5*sqrt(-T(1,1)-T(2,2)+T(3,3)+1)));

if handles.robot==1 %Si se envia al robot 1 (de incisión)
    fwrite(handles.tc_1, posicion);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    fwrite(handles.tc_1, cuaternio1);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    fwrite(handles.tc_1, cuaternio2);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    fwrite(handles.tc_1, cuaternio3);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    fwrite(handles.tc_1, cuaternio4);
elseif handles.robot==2 %Si se envia al robot 2 (de sutura)
    fwrite(handles.tc_2, posicion);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje,'Ok');
    fwrite(handles.tc_2, cuaternio1);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje,'Ok');
    fwrite(handles.tc_2, cuaternio2);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje,'Ok');
    fwrite(handles.tc_2, cuaternio3);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje,'Ok');
    fwrite(handles.tc_2, cuaternio4);
end
end
```

## Función: Obtener Coordenadas Articulares.

```
function [handles] = Obtener_Coordenadas_Articulares(hObject)
handles = guidata(hObject);
if handles.robot==1
    long_art1=str2double(char(fread(handles.tc_1,1)));
    handles.q1=str2num(char(fread(handles.tc_1,[1,long_art1])));
    long_art2=str2double(char(fread(handles.tc_1,1)));
    handles.q2=str2num(char(fread(handles.tc_1,[1,long_art2])));
    long_art3=str2double(char(fread(handles.tc_1,1)));
    handles.q3=str2num(char(fread(handles.tc_1,[1,long_art3])));
    long_art4=str2double(char(fread(handles.tc_1,1)));
    handles.q4=str2num(char(fread(handles.tc_1,[1,long_art4])));
    long_art5=str2double(char(fread(handles.tc_1,1)));
    handles.q5=str2num(char(fread(handles.tc_1,[1,long_art5])));
    long_art6=str2double(char(fread(handles.tc_1,1)));
    handles.q6=str2num(char(fread(handles.tc_1,[1,long_art6])));
elseif handles.robot==2
    long_art1=str2double(char(fread(handles.tc_2,1)));
    handles.q1=str2num(char(fread(handles.tc_2,[1,long_art1])));
    long_art2=str2double(char(fread(handles.tc_2,1)));
    handles.q2=str2num(char(fread(handles.tc_2,[1,long_art2])));
    long_art3=str2double(char(fread(handles.tc_2,1)));
    handles.q3=str2num(char(fread(handles.tc_2,[1,long_art3])));
    long_art4=str2double(char(fread(handles.tc_2,1)));
    handles.q4=str2num(char(fread(handles.tc_2,[1,long_art4])));
    long_art5=str2double(char(fread(handles.tc_2,1)));
    handles.q5=str2num(char(fread(handles.tc_2,[1,long_art5])));
    long_art6=str2double(char(fread(handles.tc_2,1)));
    handles.q6=str2num(char(fread(handles.tc_2,[1,long_art6])));
end
guidata(hObject, handles);
end
```

## Función: Incisión.

```
function [handles] = Incision(hObject)
handles = guidata(hObject);

handles.coordenadaX = str2num(handles.coordenadaX);
handles.coordenadaY = str2num(handles.coordenadaY);
handles.coordenadaZ = str2num(handles.coordenadaZ);
X = char(num2str(handles.coordenadaX));
Y = char(num2str(handles.coordenadaY));
Z = char(num2str(handles.coordenadaZ));
Zinicial = char(num2str(handles.coordenadaZ+20));
Zmedio = char(num2str(handles.coordenadaZ-handles.profundidad));
Zfinal = char(num2str(handles.coordenadaZ+80));

if handles.direccion == 2 %Si la dirección es +X

    Xfinal = char(num2str(handles.coordenadaX+handles.longitud));

    posicion_inicial=[X,',',Y,',',Zinicial];
    fwrite(handles.tc_1, posicion_inicial);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_1= [X, ',', Y, ',', Zmedio];
    fwrite(handles.tc_1, posicion_1);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_2 = [Xfinal, ',', Y, ',', Zmedio];
    fwrite(handles.tc_1, posicion_2);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_final = [Xfinal, ',', Y, ',', Zfinal];
    fwrite(handles.tc_1, posicion_final);
```

```
elseif handles.direccion == 3 %Si la dirección es 45° +X

    Xfinal = char(num2str(handles.coordenadaX+(handles.longitud*cosd(45))));
    Yfinal = char(num2str(handles.coordenadaY+(handles.longitud*sind(45))));

    posicion_inicial=[X,',',Y,',',Zinicial];
    fwrite(handles.tc_1, posicion_inicial);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_1= [X, ',', Y, ',', Zmedio];
    fwrite(handles.tc_1, posicion_1);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_2 = [Xfinal, ',', Yfinal, ',', Zmedio];
    fwrite(handles.tc_1, posicion_2);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_final = [Xfinal, ',', Yfinal, ',', Zfinal];
    fwrite(handles.tc_1, posicion_final);

elseif handles.direccion == 4 %Si la dirección es +Y

    Yfinal = char(num2str(handles.coordenadaY+handles.longitud));

    posicion_inicial=[X,',',Y,',',Zinicial];
    fwrite(handles.tc_1, posicion_inicial);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_1= [X, ',', Y, ',', Zmedio];
    fwrite(handles.tc_1, posicion_1);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_2 = [X, ',', Yfinal, ',', Zmedio];
    fwrite(handles.tc_1, posicion_2);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_final = [X, ',', Yfinal, ',', Zfinal];
    fwrite(handles.tc_1, posicion_final);

elseif handles.direccion == 5 %Si la dirección es 45° +Y

    Xfinal = char(num2str(handles.coordenadaX+(handles.longitud*sind(45))));
    Yfinal = char(num2str(handles.coordenadaY+(handles.longitud*cosd(45))));

    posicion_inicial=[X,',',Y,',',Zinicial];
    fwrite(handles.tc_1, posicion_inicial);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_1= [X, ',', Y, ',', Zmedio];
    fwrite(handles.tc_1, posicion_1);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_2 = [Xfinal, ',', Yfinal, ',', Zmedio];
    fwrite(handles.tc_1, posicion_2);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_final = [Xfinal, ',', Yfinal, ',', Zfinal];
    fwrite(handles.tc_1, posicion_final);

elseif handles.direccion == 6 %Si la dirección es -X

    Xfinal = char(num2str(handles.coordenadaX-handles.longitud));

    posicion_inicial=[X,',',Y,',',Zinicial];
    fwrite(handles.tc_1, posicion_inicial);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
    posicion_1= [X, ',', Y, ',', Zmedio];
    fwrite(handles.tc_1, posicion_1);
    mensaje=fread(handles.tc_1,2);
    waitfor(mensaje,'Ok');
```

```

posicion_2 = [Xfinal, ',', Y, ',', Zmedio];
fwrite(handles.tc_1, posicion_2);
mensaje=fread(handles.tc_1,2);
waitfor(mensaje,'Ok');
posicion_final = [Xfinal, ',', Y, ',', Zfinal];
fwrite(handles.tc_1, posicion_final);

elseif handles.direccion == 7 %Si la dirección es 45° -X

Xfinal = char(num2str(handles.coordenadaX-(handles.longitud*cosd(45))));
Yfinal = char(num2str(handles.coordenadaY-(handles.longitud*sind(45))));

posicion_inicial=[X,',',Y,',',Zinicial];
fwrite(handles.tc_1, posicion_inicial);
mensaje=fread(handles.tc_1,2);
waitfor(mensaje,'Ok');
posicion_1= [X, ',', Y, ',', Zmedio];
fwrite(handles.tc_1, posicion_1);
mensaje=fread(handles.tc_1,2);
waitfor(mensaje,'Ok');
posicion_2 = [Xfinal, ',', Yfinal, ',', Zmedio];
fwrite(handles.tc_1, posicion_2);
mensaje=fread(handles.tc_1,2);
waitfor(mensaje,'Ok');
posicion_final = [Xfinal, ',', Yfinal, ',', Zfinal];
fwrite(handles.tc_1, posicion_final);

elseif handles.direccion == 8 %Si la dirección es -Y

Yfinal = char(num2str(handles.coordenadaY-handles.longitud));

posicion_inicial=[X,',',Y,',',Zinicial];
fwrite(handles.tc_1, posicion_inicial);
mensaje=fread(handles.tc_1,2);
waitfor(mensaje,'Ok');
posicion_1= [X, ',', Y, ',', Zmedio];
fwrite(handles.tc_1, posicion_1);
mensaje=fread(handles.tc_1,2);
waitfor(mensaje,'Ok');
posicion_2 = [X, ',', Yfinal, ',', Zmedio];
fwrite(handles.tc_1, posicion_2);
mensaje=fread(handles.tc_1,2);
waitfor(mensaje,'Ok');
posicion_final = [X, ',', Yfinal, ',', Zfinal];
fwrite(handles.tc_1, posicion_final);

elseif handles.direccion == 9 %Si la dirección es 45° -Y

Xfinal = char(num2str(handles.coordenadaX-(handles.longitud*sind(45))));
Yfinal = char(num2str(handles.coordenadaY-(handles.longitud*cosd(45))));

posicion_inicial=[X,',',Y,',',Zinicial];
fwrite(handles.tc_1, posicion_inicial);
mensaje=fread(handles.tc_1,2);
waitfor(mensaje,'Ok');
posicion_1= [X, ',', Y, ',', Zmedio];
fwrite(handles.tc_1, posicion_1);
mensaje=fread(handles.tc_1,2);
waitfor(mensaje,'Ok');
posicion_2 = [Xfinal, ',', Yfinal, ',', Zmedio];
fwrite(handles.tc_1, posicion_2);
mensaje=fread(handles.tc_1,2);
waitfor(mensaje,'Ok');
posicion_final = [Xfinal, ',', Yfinal, ',', Zfinal];
fwrite(handles.tc_1, posicion_final);

end
end

```

## Función: Sutura.

```
function [handles] = Sutura(hObject)
handles = guidata(hObject);
global longitud direccion coordenadaX coordenadaY coordenadaZ;
handles.distancia_grapas = longitud/handles.grapas;

for i=1:handles.grapas

    X = char(num2str(-coordenadaX+1760));
    Y = char(num2str(-coordenadaY-500));
    Z = char(num2str(coordenadaZ));
    Zarriba = char(num2str(coordenadaZ+50));

    if direccion == 2 %Si la dirección es +X

        X = char(num2str(-coordenadaX+1760-((i-1)*handles.distancia_grapas)));

        posicion_inicial=[X, ',', Y, ',', Zarriba];
        fwrite(handles.tc_2, posicion_inicial);
        mensaje=fread(handles.tc_2,2);
        waitfor(mensaje, 'Ok');
        posicion_medio=[X, ',', Y, ',', Z];
        fwrite(handles.tc_2, posicion_medio);
        mensaje=fread(handles.tc_2,2);
        waitfor(mensaje, 'Ok');
        posicion_final=[X, ',', Y, ',', Zarriba];
        fwrite(handles.tc_2, posicion_final);
        mensaje=fread(handles.tc_2,2);
        waitfor(mensaje, 'Ok');

    elseif direccion == 3 %Si la dirección es 45° +X

        X = char(num2str(-coordenadaX+1760-((i-1)*handles.distancia_grapas*cosd(45))));
        Y = char(num2str(-coordenadaY-500-((i-1)*handles.distancia_grapas*sind(45))));

        posicion_inicial=[X, ',', Y, ',', Zarriba];
        fwrite(handles.tc_2, posicion_inicial);
        mensaje=fread(handles.tc_2,2);
        waitfor(mensaje, 'Ok');
        posicion_medio=[X, ',', Y, ',', Z];
        fwrite(handles.tc_2, posicion_medio);
        mensaje=fread(handles.tc_2,2);
        waitfor(mensaje, 'Ok');
        posicion_final=[X, ',', Y, ',', Zarriba];
        fwrite(handles.tc_2, posicion_final);
        mensaje=fread(handles.tc_2,2);
        waitfor(mensaje, 'Ok');

    elseif direccion == 4 %Si la dirección es +Y

        Y = char(num2str(-coordenadaY-500-((i-1)*handles.distancia_grapas)));

        posicion_inicial=[X, ',', Y, ',', Zarriba];
        fwrite(handles.tc_2, posicion_inicial);
        mensaje=fread(handles.tc_2,2);
        waitfor(mensaje, 'Ok');
        posicion_medio=[X, ',', Y, ',', Z];
        fwrite(handles.tc_2, posicion_medio);
        mensaje=fread(handles.tc_2,2);
        waitfor(mensaje, 'Ok');
        posicion_final=[X, ',', Y, ',', Zarriba];
        fwrite(handles.tc_2, posicion_final);
        mensaje=fread(handles.tc_2,2);
        waitfor(mensaje, 'Ok');
```



```
elseif direccion == 5 %Si la dirección es 45° +Y

    X = char(num2str(-coordenadaX+1760+((i-1)*handles.distancia_grapas*sind(45)));
    Y = char(num2str(-coordenadaY-500-((i-1)*handles.distancia_grapas*cosd(45)));

    posicion_inicial=[X, ',', Y, ',', Zarriba];
    fwrite(handles.tc_2, posicion_inicial);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje,'Ok');
    posicion_medio=[X, ',', Y, ',', Z];
    fwrite(handles.tc_2, posicion_medio);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje,'Ok');
    posicion_final=[X, ',', Y, ',', Zarriba];
    fwrite(handles.tc_2, posicion_final);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje,'Ok');

elseif direccion == 6 %Si la dirección es -X

    X = char(num2str(-coordenadaX+1760+((i-1)*handles.distancia_grapas)));

    posicion_inicial=[X, ',', Y, ',', Zarriba];
    fwrite(handles.tc_2, posicion_inicial);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje,'Ok');
    posicion_medio=[X, ',', Y, ',', Z];
    fwrite(handles.tc_2, posicion_medio);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje,'Ok');
    posicion_final=[X, ',', Y, ',', Zarriba];
    fwrite(handles.tc_2, posicion_final);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje,'Ok');

elseif direccion == 7 %Si la dirección es 45° -X

    X = char(num2str(-coordenadaX+1760+((i-1)*handles.distancia_grapas*cosd(45)));
    Y = char(num2str(-coordenadaY-500+((i-1)*handles.distancia_grapas*sind(45)));

    posicion_inicial=[X, ',', Y, ',', Zarriba];
    fwrite(handles.tc_2, posicion_inicial);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje,'Ok');
    posicion_medio=[X, ',', Y, ',', Z];
    fwrite(handles.tc_2, posicion_medio);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje,'Ok');
    posicion_final=[X, ',', Y, ',', Zarriba];
    fwrite(handles.tc_2, posicion_final);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje,'Ok');

elseif direccion == 8 %Si la dirección es -Y

    Y = char(num2str(-coordenadaY-500+((i-1)*handles.distancia_grapas)));

    posicion_inicial=[X, ',', Y, ',', Zarriba];
    fwrite(handles.tc_2, posicion_inicial);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje,'Ok');
    posicion_medio=[X, ',', Y, ',', Z];
    fwrite(handles.tc_2, posicion_medio);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje,'Ok');
    posicion_final=[X, ',', Y, ',', Zarriba];
    fwrite(handles.tc_2, posicion_final);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje,'Ok');
```





```
elseif direccion == 9 %Si la dirección es 45° -Y

    X = char(num2str(-coordenadaX+1760-((i-1)*handles.distancia_grapas*sind(45)));
    Y = char(num2str(-coordenadaY-500+((i-1)*handles.distancia_grapas*cosd(45)));

    posicion_inicial=[X, ',', Y, ',', Zarriba];
    fwrite(handles.tc_2, posicion_inicial);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje, 'Ok');
    posicion_medio=[X, ',', Y, ',', Z];
    fwrite(handles.tc_2, posicion_medio);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje, 'Ok');
    posicion_final=[X, ',', Y, ',', Zarriba];
    fwrite(handles.tc_2, posicion_final);
    mensaje=fread(handles.tc_2,2);
    waitfor(mensaje, 'Ok');

end
end
end
```

## ANEXO IV. SCRIPTS DE RAPID ROBOTSTUDIO.

### Función: Enviar Posición.

```
MODULE Enviar_Posicion
  PROC EnviarPosicion ()
    posicion:=CPos(\Tool:=Herramienta_Incision_TCP \WObj:=wobj0);
    posicionX:=ValToStr(posicion.x);
    posicionY:=ValToStr(posicion.y);
    posicionZ:=ValToStr(posicion.z);
    longitudX:=ValToStr(StrLen(posicionX));
    longitudY:=ValToStr(StrLen(posicionY));
    longitudZ:=ValToStr(StrLen(posicionZ));
    SocketSend cliente, \Str:=longitudX;
    SocketSend cliente, \Str:=longitudY;
    SocketSend cliente, \Str:=longitudZ;
    SocketSend cliente, \Str:=posicionX;
    SocketSend cliente, \Str:=posicionY;
    SocketSend cliente, \Str:=posicionZ;
  ENDPROC
ENDMODULE
```

### Función: Recibir Posición.

```
MODULE Recibir_Posicion
  PROC RecibirPosicion ()
    SocketReceive cliente, \Str:=coordenadas \Time:=20;

    Pos1:=StrFind(coordenadas,1,"");
    Pos2:=StrFind(coordenadas,Pos1+1,"");
    Pos3:=StrLen(coordenadas);

    Ok:=StrToVal(StrPart(coordenadas,1,Pos1-1),x);
    Ok:=StrToVal(StrPart(coordenadas,Pos1+1,Pos2-Pos1-1),y);
    Ok:=StrToVal(StrPart(coordenadas,Pos2+1,Pos3-Pos2),z);

    posicion:=[x,y,z];
  ENDPROC
ENDMODULE
```

### Función: Mover Robot.

```
MODULE Mover_Robot
  PROC MoverRobot ()
    IF movimiento="1" THEN !Si el movimiento es en coordenadas articulares
      SocketReceive cliente, \Str:=coordenadas \Time:=20;
      Pos1:=StrFind(coordenadas,1,"");
      Pos2:=StrFind(coordenadas,Pos1+1,"");
      Pos3:=StrLen(coordenadas);

      Ok:=StrToVal(StrPart(coordenadas,1,Pos1-1),x);
      Ok:=StrToVal(StrPart(coordenadas,Pos1+1,Pos2-Pos1-1),y);
      Ok:=StrToVal(StrPart(coordenadas,Pos2+1,Pos3-Pos2),z);
    
```

```

posicion:=[x,y,z];
SocketSend cliente, \Str:=comprobacion;
SocketReceive cliente, \Str:=q1 \Time:=20;
Ok:=StrToVal(q1,q_1);
SocketSend cliente, \Str:=comprobacion;
SocketReceive cliente, \Str:=q2 \Time:=20;
Ok:=StrToVal(q2,q_2);
SocketSend cliente, \Str:=comprobacion;
SocketReceive cliente, \Str:=q3 \Time:=20;
Ok:=StrToVal(q3,q_3);
SocketSend cliente, \Str:=comprobacion;
SocketReceive cliente, \Str:=q4 \Time:=20;
Ok:=StrToVal(q4,q_4);

MoveJ [posicion,[q_1,q_2,q_3,q_4],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
vel,precision,tool0\WObj:=wobj0;

ELSEIF movimiento="2" THEN !Si el movimiento es en coordenadas cartesianas
SocketReceive cliente, \Str:=coordenadas \Time:=20;

Pos1:=StrFind(coordenadas,1,"");
Pos2:=StrFind(coordenadas,Pos1+1,"");
Pos3:=StrLen(coordenadas);

Ok:=StrToVal(StrPart(coordenadas,1,Pos1-1),x);
Ok:=StrToVal(StrPart(coordenadas,Pos1+1,Pos2-Pos1-1),y);
Ok:=StrToVal(StrPart(coordenadas,Pos2+1,Pos3-Pos2-1),z);

posicion:=[x,y,z];
posicion_articulaciones:=[posicion,[q_1,q_2,q_3,q_4],[0,0,0,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
MoveJ posicion_articulaciones,vel,precision,tool0\WObj:=wobj0; !Se mueve con la misma
orientacion que el movimiento anterior
articulaciones:=CalcJointT(posicion_articulaciones,tool0\WObj:=wobj0);
art_1:=ValToStr(Round(articulaciones.robax.rax_1 \Dec:=5));
long_art1:=ValToStr(StrLen(art_1));
art_2:=ValToStr(Round(articulaciones.robax.rax_2 \Dec:=5));
long_art2:=ValToStr(StrLen(art_2));
art_3:=ValToStr(Round(articulaciones.robax.rax_3 \Dec:=5));
long_art3:=ValToStr(StrLen(art_3));
art_4:=ValToStr(Round(articulaciones.robax.rax_4 \Dec:=5));
long_art4:=ValToStr(StrLen(art_4));
art_5:=ValToStr(Round(articulaciones.robax.rax_5 \Dec:=5));
long_art5:=ValToStr(StrLen(art_5));
art_6:=ValToStr(Round(articulaciones.robax.rax_6 \Dec:=5));
long_art6:=ValToStr(StrLen(art_6));
SocketSend cliente \Str:=long_art1; !Enviar todas los valores de las articulaciones para actualizar
valores y poder mover en coordenadas articulares
SocketSend cliente \Str:=art_1;
SocketSend cliente \Str:=long_art2;
SocketSend cliente \Str:=art_2;
SocketSend cliente \Str:=long_art3;
SocketSend cliente \Str:=art_3;
SocketSend cliente \Str:=long_art4;

```



```
SocketSend cliente \Str:=art_4;  
SocketSend cliente \Str:=long_art5;  
SocketSend cliente \Str:=art_5;  
SocketSend cliente \Str:=long_art6;  
SocketSend cliente \Str:=art_6;  
  
ENDIF  
ENDPROC  
ENDMODULE
```

## Función: Funcionalidad de la herramienta de sutura.

```
IF mensaje="Girar pinza" THEN  
    SocketReceive cliente, \Str:=giro, \Time:=15;  
    SETDO AccionNormal,1; !Inicializamos los valores de las acciones  
    SETDO Accion45,0;  
    SETDO Accion90,0;  
    SETDO Accion135,0;  
    IF giro="Pinza 0" THEN  
        SETDO AccionNormal,0;  
    ELSEIF giro="Pinza 45" THEN  
        SETDO Accion45,1;  
    ELSEIF giro="Pinza 90" THEN  
        SETDO Accion90,1;  
    ELSEIF giro="Pinza 135" THEN  
        SETDO Accion135,1;  
    ENDIF  
  
ELSEIF mensaje="Apertura pinza" THEN  
    SocketReceive cliente, \Str:=apertura, \Time:=15;  
    IF apertura="Abrir pinza" THEN  
        IF giro="Pinza 0" THEN  
            SETDO AccionNormal,0;  
        ELSEIF giro="Pinza 45" THEN  
            SETDO Accion45,1;  
        ELSEIF giro="Pinza 90" THEN  
            SETDO Accion90,1;  
        ELSEIF giro="Pinza 135" THEN  
            SETDO Accion135,1;  
        ENDIF  
    ELSEIF apertura="Cerrar pinza" THEN  
        IF giro="Pinza 0" THEN  
            SETDO AccionNormal,1;  
        ELSEIF giro="Pinza 45" THEN  
            SETDO Accion45,0;  
        ELSEIF giro="Pinza 90" THEN  
            SETDO Accion90,0;  
        ELSEIF giro="Pinza 135" THEN  
            SETDO Accion135,0;  
        ENDIF  
    ENDIF  
ENDIF  
ENDIF
```